INSTITUTO TECNOLÓGICO
Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación el 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

DOCTORADO EN CIENCIAS DE LA INGENIERÍA

DISEÑO E IMPLEMENTACIÓN DE UNIDADES DE RAÍZ CUADRADA INVERSA EN TECNOLOGÍA ASIC DIGITAL PARA APLICACIONES EMBEBIDAS DE BAJA POTENCIA

Tesis que para obtener el grado de

DOCTOR EN CIENCIAS DE LA INGENIERÍA

presenta: Cuauhtémoc Rafael Aguilera Galicia

Director de tesis: Dr. Omar Humberto Longoria Gándara

Co-director de tesis: Dr. José Luis Pizano Escalante

Tlaquepaque, Jalisco. Marzo de 2019

DOCTOR EN CIENCIAS DE LA INGENIERÍA (2019) ITESO, Tlaquepaque, Jal., México

TÍTULO: **Diseño e implementación de unidades de raíz cuadrada inversa en tecnología ASIC digital para aplicaciones embebidas de baja potencia**

AUTOR: Cuauhtémoc Rafael Aguilera Galicia
Ingeniero en Telecomunicaciones y Electrónica (Universidad de Guanajuato, México)
Maestro en Diseño Electrónico (ITESO, México)

DIRECTOR DE TESIS: Omar Humberto Longoria Gándara
Departamento de Electrónica, Sistemas e Informática, ITESO
Ingeniero en Electrónica y Comunicaciones (ITESM, Campus Monterrey, México)
Maestro en Ciencias de Ingeniería Eléctrica, opción Telecomunicaciones (CINVESTAV, México)
Doctor en Ciencias de Ingeniería Eléctrica, opción Telecomunicaciones (CINVESTAV, México)

NÚMERO DE PÁGINAS: xxiii, 131

ITESO – The Jesuit University of Guadalajara

Department of Electronics, Systems, and Informatics

DOCTORAL PROGRAM IN ENGINEERING SCIENCES

DESIGN AND IMPLEMENTATION OF RECIPROCAL SQUARE ROOT UNITS ON DIGITAL ASIC TECHNOLOGY FOR LOW POWER EMBEDDED APPLICATIONS

Thesis to obtain the degree of

DOCTOR IN ENGINEERING SCIENCES

Presents: Cuauhtémoc Rafael Aguilera-Galicia

Thesis Director: Dr. Omar Humberto Longoria-Gandara

Thesis Co-director: Dr. José Luis Pizano-Escalante

Tlaquepaque, Jalisco, Mexico
March 2019

PHD IN ENGINEERING SCIENCES (2019) ITESO, Tlaquepaque, Jal., Mexico

TITLE: **Design and Implementation of Reciprocal Square Root Units on Digital ASIC Technology for Low Power Embedded Applications**

AUTHOR: Cuauhtémoc Rafael Aguilera-Galicia
Bachelor's degree in telecommunications and electronics engineering (University of Guanajuato, Mexico)
Master's degree in electrical engineering (ITESO, Mexico)

THESIS DIRECTOR: Omar Humberto Longoria-Gándara
Department of Electronics, Systems, and Informatics, ITESO
Bachelor's degree in electronics engineering (ITESM, Mexico)
Master's degree in electrical engineering (CINVESTAV, Mexico)
Ph.D. degree in electrical engineering (CINVESTAV, Mexico)

NUMBER OF PAGES: xxiii, 131

To all humans who want to make this world a better place for all living beings.

Resumen

Aplicaciones emergentes tales como el internet de las cosas (IoT, por sus siglas en inglés), ciudades inteligentes, y vehículos autónomos, demandan sistemas electrónicos más eficientes y más pequeños. Esto impone la necesidad de desarrollar dichos sistemas con bajo consumo de energía, tamaño reducido, y tiempos de desarrollo cortos para un mercado mundial. En este escenario, el desarrollo de sistemas en circuito integrado (SOC, por sus siglas en inglés) es una solución atractiva y viable, ya que las tecnologías SOC permiten realizar diseños a la medida, en tecnologías nanométricas, y con arquitecturas y tecnologías para bajo consumo de potencia. Un SOC se desarrolla con la integración de múltiples módulos pequeños previamente diseñados y verificados, los cuales son conocidos como propiedades intelectuales de semiconductor o núcleos IP. Esta tesis doctoral plantea el diseño e implementación de núcleos IP implementados en tecnologías de circuito integrado de aplicación específica (ASIC, por sus siglas en inglés). En particular se presenta el diseño e implementación de dos núcleos IP para calcular el recíproco de la raíz cuadrada (RSR, por sus siglas en inglés). La operación RSR es una de las operaciones fundamentales más complejas; su ejecución requiere varios ciclos de reloj y es altamente demandante de recursos de hardware. Esta operación es utilizada en sistemas de comunicaciones inalámbricas, renderizado de imágenes en sistemas multimedia, entre otras aplicaciones. El algoritmo implementado para calcular la operación RSR está basado en el método de Newton-Raphson, donde la semilla es proporcionada por una aproximación polinomial por partes. El primer núcleo IP propuesto, 2C-RSR, utiliza aritmética de punto fijo con 16 bits, el cual fue manufacturado por MOSIS en tecnología ASIC CMOS de 130 nm. Mediciones del circuito integrado propuesto son comparadas con las de diseños existentes; los resultados muestran que las características de baja latencia y bajo consumo de potencia de la IP 2C-RSR, son adecuadas para aplicaciones en sistemas embebidos de bajo consumo de potencia y bajo costo computacional. El segundo núcleo IP propuesto, HF-2cRSR, también calcula la operación RSR y utiliza aritmética de punto flotante de media precisión (FP16); este formato está definido en el estándar 754-2008 del IEEE. La IP HF-2cRSR fue implementada en dos tecnologías FPGA con el propósito de ser comparado con núcleos IP comerciales de Intel y Xilinx. Los resultados muestran las ventajas de la baja latencia del HF-2cRSR en su rendimiento, y el impacto del formato FP16 en los recursos utilizados.

Summary

Current and emerging applications of information and communication technologies (ICT), such as the internet of things (IoT), smart cities, autonomous vehicles, among others, demand more efficient and smaller size electronic and computer systems. This imposes challenges to electronic designers, to create the corresponding systems with low-power consumption, small size, and short development times for a world market. In this scenario, the development of systems on chips (SOC) is an attractive and viable solution because SOC technologies allow tailored designs in nanometric technologies, and with architectures and technologies for low-power consumption. A SOC is developed by the integration of multiple small modules previously designed and verified. These modules are known as semiconductor intellectual properties or IP cores. This doctoral dissertation proposes the design and implementation of IP cores implemented on ASIC technology. Specifically, this document presents the design and implementation of two IP cores to calculate the reciprocal of the square root (RSR). The RSR operation is one of the most complex fundamental operations; its execution requires several clock cycles and it is highly demanding of hardware resources. This operation is used in wireless communication systems, images rendering for multimedia systems, among other applications. The implemented algorithm to calculate the RSR operation is based on the Newton-Raphson method, were the seed is provided by a piecewise-polynomial approximation. The first proposed IP core, 2C-RSR, uses 16-bit fixed-point arithmetic. The 2C-RSR was manufactured by MOSIS on 130 nm CMOS ASIC technology. Experimental measurements of the proposed integrated circuit are compared with corresponding existing designs; the results show that the low latency and low-power consumption characteristics of the 2C-RSR are suitable for low power and low-computational cost embedded-system applications. The second proposed IP core, HF-2cRSR, also calculates the RSR operation and it uses half-precision floating-point (FP16) arithmetic to perform the computation. This arithmetic format is defined by the IEEE 754-2008 standard. In addition to performing the logic synthesis of the HF-2cRSR on ASIC technology, it was also implemented on two FPGA technologies with the purpose of comparing with commercial IP cores from Intel and Xilinx. The results show the advantages of the HF-2cRSR low latency on its throughput, and the impact of the FP16 format on the utilized resources.

Acknowledgements

The author wishes to express his sincere appreciation to Dr. Omar Humberto Longoria-Gándara, professor of the Department of Electronics, Systems, and Informatics at ITESO, for his encouragement, expert guidance and keen supervision as doctoral thesis director throughout the course of this work. The author offers his gratitude to Dr. José Luis Pizano-Escalante, from the Department of Electronics, Systems, and Informatics at ITESO, for his valuable contributions as doctoral thesis co-director during the development of this work. He also thanks to Dr. Manuel Salim-Maza, Dr. Víctor Avendaño-Fernández, and Dr. Javier Vázquez-Castillo, members of his Ph.D. Thesis Committee, for their interest, assessment, and suggestions.

It is the author's pleasure to express his thankfulness to Dr. José Ernesto Rayas-Sánchez for his outstanding labor as Chair of this doctoral program, and his diligent and patient help with the development of my internal research reports, and this thesis document.

The author is grateful to Dr. Dong S. Ha, professor of Virginia Tech for making the VTVT 180 nm standard-cell library available for this work.

Special thanks are due to Joaquín García, Oscar Guzmán, and Alfredo Delgado from Intel Design Center of Guadalajara, for their decisive cooperation and help for testing the manufactured integrated circuit developed in this doctoral dissertation.

The author thanks to Diego Armando Hernández-Ramírez for his valuable help to the Cadence CAD tools set up.

The author gratefully acknowledges the authorities of ITESO for the financial support through an assistantship granted by the ITESO's Program for Academic Level Enhancement (*Programa de Superación del Nivel Académico, PSNA*).

Finally, special thanks are due to my family and friends: my son Jonathan Rafael, my parents Gloria Galicia and Rafael Aguilera, my sisters Lilia, Chelito, and Minerva, my brothers Benito, and Salvador. My friends Joaquín Fortun, Miriam Galindo, and José María Valencia for their help, understanding, and continuous loving support.

Contenido

Contents

List of Figures

List of Tables

Introduction

The increasing use of mobile computing platforms in commercial and everyday human activities, demands more efficiency, smaller size, and higher functionality in the corresponding electronic systems. Communication and multimedia technologies are essential in these systems which are supported by digital signal processing (DSP) techniques. Therefore, there is a need for DSP modules implemented in VLSI hardware that enable circuits with the performance, the silicon area and the power consumption suitable for the specific application.

There are DSP algorithms of extensive and frequent use, which require intensive computation that is performed commonly by a general-purpose microprocessor, making this the processing system bottleneck. This type of algorithms could be performed in tailored VLSI hardware with the aim of improving overall system performance in terms of speed, silicon area, and power consumption. After implementing the digital signal processing algorithms in VLSI circuits, these could be integrated into a system on a chip (SOC) through an intellectual-property (IP) instantiation. Semiconductor intellectual properties, also known as IP cores, are proven and reusable units, which can be implemented at different abstraction levels: generic logic, technology cells or chip layout. They are classified on soft, firm, and hard IP cores and are part of a growing trend in the electronic design industry because reducing design time (time-to-market) and could improve overall system performance.

Two of the main electronic design challenges that current and future applications impose, such as IoT and deep learning, are low-power consumption and system design based on modular reusable components [Blaauw-14]. In front of this scenario, IP cores are essential elements of design reuse and tailored design to achieve the requirements of silicon area and power consumption.

An emerging approach to save energy in electronic systems design is approximate computing [Ho-17], which is useful for many applications that are tolerant to low accuracy. For example, in the deep-learning field it has been demonstrated that a neural-network accelerator can be trained and implemented using half-precision floating-point (FP16) arithmetic [Venkatesh-17], achieving high accuracy and performance in image classification, while reducing power-consumption and computational requirements. For these kinds of applications, it is advantageous

to trade off precision for gain in efficiency and performance. For example, in [Mittal-16] is shown that admitting only 5% of classification-accuracy loss in the k-means clustering algorithm, produces 50 times energy saving compared with respect to the fully accurate algorithm.

The reciprocal of the square root (RSR) is one of the most complex fundamental operations, it demands many hardware resources and requires several clock cycles to be executed by a sequential microprocessor. For instance, the RSR operation requires from 6 to 10 clock cycles on Intel 64-bit architectures and approximately 60 clock cycles on an embedded microprocessor such as the ARM Cortex-M4. The RSR operation is essential in many DSP algorithms where matrix decomposition techniques are required for the solution of systems of linear equation. For example, the singular value decomposition (SVD), which is applied in wireless communication systems for modulation techniques such as the Orthogonal Frequency Division Multiplexing (OFDM); The Cholesky decomposition for channel estimation; and the Gram-Schmidt QR decompositions for matrix inversion. Furthermore, the RSR operation is applied in gaming for 3D-image rendering.

The motivation of this research work is the development of silicon IP cores for low-power embedded applications. In this doctoral dissertation, is presented the design and implementation of two arithmetic IP cores for computing the RSR operation. Both IP cores are based on the Newton-Raphson algorithm, where the seed is provided by a piecewise-polynomial approximation [Pizano-Escalante-15]. Several RSR implementations have been reported, however, they are mainly focused on accelerating double-precision floating-point (FP64) units [Ercegovac-00], [Piñeiro-02], or single-precision floating-point (FP32) units [Wires-06], [Kwon-08], [Suresh-13], which are not suitable for low-power embedded applications. To this kind of applications, fixed-point (FxP) arithmetic is preferred since utilize fewer hardware resources and lees power consumption than floating-point (FP) arithmetic.

The first proposed implementation, 2C-RSR, is a hard-IP core implemented on an ASIC CMOS technology of 130 nm. It computes the RSR of a 16-bit FxP number. The 2C-RSR IP core was prototyped by MOSIS[1], the chip produces a new result in only two clock cycles and all the results are bit-accurate. Experimental measurements of the 2C-RSR chip show that its power consumption is several times lower than previously published firm-IP cores, which are synthesized designs on standard-cell technologies. Since the 2C-RSR IP-core exhibits the lowest latency with respect to the compared implementations, it produces higher throughput at common working

[1] The MOSIS Service, What is MOSIS. Jan. 24, 2019, https://www.mosis.com/what-is-mosis.

frequencies of low-power embedded applications. These characteristics make the proposed chip a useful silicon intellectual property, suitable for embedded applications where low power, low latency, and low hardware cost are required.

The second proposed implementation, HF-2cRSR, is a firm-IP Core for computing the RSR operation using floating-point arithmetic, it exploits the characteristics of the half-precision floating-point IEEE 754-2008 standard, which offers higher dynamic range than 16-bit FxP format and utilizes fewer hardware resources than FP64 and FP32 formats. This doctoral dissertation documents the design and verification of the HF-2cRSR Verilog model, and exemplify the design of FP-arithmetic modules through the detailed design of a tailored FP multiplier for the HF-2cRSR. The RTL model of the HF-2cRSR IP-core is synthesized on ARM 130 nm standard-cell CMOS technology and on FPGA technology. In order to compare the HF-2cRSR with respect to commercial IP-cores, it is implemented on two FPGAs from Intel and Xilinx. The implementation results show that the HF-2cRSR IP-core meets the error specification defined by the IEEE 754-2008 standard. The maximum relative error is 4.8768532×10^{-4}, which is lower than ½ ulp of the half-precision format. The advantage of the FP16 arithmetic over the corresponding FP32 is observed on the utilized resources, for example, the multipliers sizes. Both commercial designs present higher working frequency, however, the HF-2cRSR IP-core, exhibits the lowest latency, of only 2 clock cycles. For this reason, the proposed implementation offers 40% more throughput than Xilinx IP and 72% more than Intel IP core. These characteristics make the HF-2cRSR IP-core adequate for low-power embedded applications.

In order to design and send to manufacture the proposed IP core in this doctoral dissertation, CAD tools, a CMOS process design kit (PDK), and the corresponding standard-cell libraries are selected. This allowed making the setup of the ITESO integrated circuit laboratory and implemented for the first time a digital ASIC design flow.

This doctoral dissertation is organized as follows:

In Chapter 1, the digital ASIC design flow is presented and its front-end and back-end stages are described. Several Cadence® tools to perform this design flow are presented and a subset of these tools to implement the digital ASIC design flow at ITESO is recommended. The Cadence's workflow for the logic synthesis of an ASIC, which is the main step of the front-end stage is presented in detail.

Chapter 2 describes some tasks to implement a digital ASIC design flow at ITESO

integrated circuit laboratory. The selection of a PDK for prototyping digital ASIC through a MOSIS research program is presented. The fundamental components to perform the digital ASIC design are identified and described. The logic synthesis of a basic circuit using the Virginia Tech standard-cells and Cadence tool is performed, the workflow, input/output elements and the synthesis results are presented.

In Chapter 3, the design and on-chip implementation of a hard-IP core are presented. This IP core computes the RSR operation utilizing FxP arithmetic. After introduced the relevance of the RSR operation in DSP applications, some related works are commented. The implemented algorithm is discussed, and its architecture is explained. The ASIC physical design is presented, and the experimental measurement results are reported, as well as the comparison with respect to existing standard-cell based designs.

Chapter 4 documents the design and implementation of a half-precision floating-point RSR IP-core. An introduction to the half-precision format and the specifications to the floating-point RSR operation are presented. The proposed architecture is discussed, and its logic synthesis results are reported. The FPGA implementation is performed and the comparison with respect to commercial IP cores from Intel and Xilinx is reported.

This thesis document includes four appendixes. Appendix A presents a reference list of the internal research reports that were written as part of my doctoral studies. Appendix B shows the reference list of journal and conference papers published during my doctoral studies. Appendix C reports a glossary of common acronyms used in Cadence documents about ASIC design tools. Finally, Appendix D shows a script example to perform ASIC logic synthesis using Cadence RTL Compiler.

1. Digital ASIC Design Flow and Cadence Tools

In this chapter, the general digital ASIC design flow is described and some Cadence® tools are presented, a specific selection of which is proposed to be installed in the ITESO integrated circuit laboratory for digital ASIC designing. The logic synthesis steps are presented in detail given that the tools required for performing this stage have not yet been installed in the ITESO integrated circuits laboratory and it is of first importance for having the capability to perform the full digital ASIC design flow.

ITESO plans on having the capacity of designing digital ASIC's, which can be implemented using nanometric CMOS standard-cell technology and manufactured by fab's with which the ITESO has signed an agreement. These designs and circuits will be produced for educational and research purposes.

Cadence®, a world leader in EDA tools for designing electronic systems on a single chip (SoC), offers an enormous variety of products[2, 3]. Cadence tools systematize the ASIC development cycle, and verify the different phases, ranging from the concept and system modeling to packaging.

The wide range of sophisticated Cadence tools makes selecting the appropriate versions to be installed and maintained by ITESO personnel particularly important. This will allow us to move towards the goal of implementing a design flow of digital ASIC's at ITESO, with the tools and design kits that enable us to design, verify, and send to manufacture digital integrated circuits.

This chapter is a brief presentation of general digital ASIC design flow, emphasizing the phases that we consider basic aspects of the process and are required to learn at ITESO. In addition, the appropriate Cadence tools are selected for each stage of the design cycle and its implementation at ITESO.

[2] Cadence-Tools, Integrated Design and Verification Technologies, Methodologies, and Application Specific Kits. Feb. 02, 2014, http://www.cadence.com/products/pages/default.aspx.

[3] Cadence-Download, Cadence Releases Available for Installation. Feb. 02, 2014, http://downloads.cadence.com/ESDWeb/ProductDetail.eo?methodToCall=viewProductsInRelease&baseReleaseName=REL%20INCISIV12.2&releaseName=INCISIV122&platform=LINUX.

1.1. Digital ASIC Design Flow

Digital ASIC design flow consists of several steps [Franzon-99], [Dharwadkar-10], which can vary depending on the design complexity and the available CAD tools. However digital ASIC design flow can be divided into two big parts: front-end design and back-end design:

i. Front-End Design includes design specification, architectural design, behavioral description, functional verification, RTL description, RTL compilation and verification, and logic synthesis.
ii. Back-End Design, also known as physical design, includes partitioning, floor-planning, placement, clock tree synthesis, signal routing and timing closure.

The logic synthesis stage consists of several steps that will be described more in detail in sections 1.1.1.1, and 1.3, given that the tools required for performing this phase have not yet been installed in the ITESO integrated circuits laboratory and we want to have the capability to carry out full digital ASIC design flow. Fig. 1.1 outlines a general digital ASIC design flow.

1.1.1 Front-End Design

Digital ASIC design starts with an informal description of the problem to be resolved based on requirement analysis and includes a list of the new ASIC's functions. The objective of this step is to write down design specifications in a complete and formal document (see Fig. 1.1).

The architectural design stage commonly uses the top-down methodology for proposing an overview of the system; sub-system parts are normally specified using black boxes. The system is subdivided several times until each building block is perfectly identified with frequently used digital modules: memories, registers, finite-state machines (FSM), arithmetic circuits, gates, etc. establishing the relationship between the parts.

In the behavioral description stage, the design team creates a functional model of the system in a high-level language, which is very useful for verifying specifications, validating the ASIC functions suitability and developing test vectors used in subsequent verification stages. The behavioral model must be simulated for verifying that functionality meets specifications.

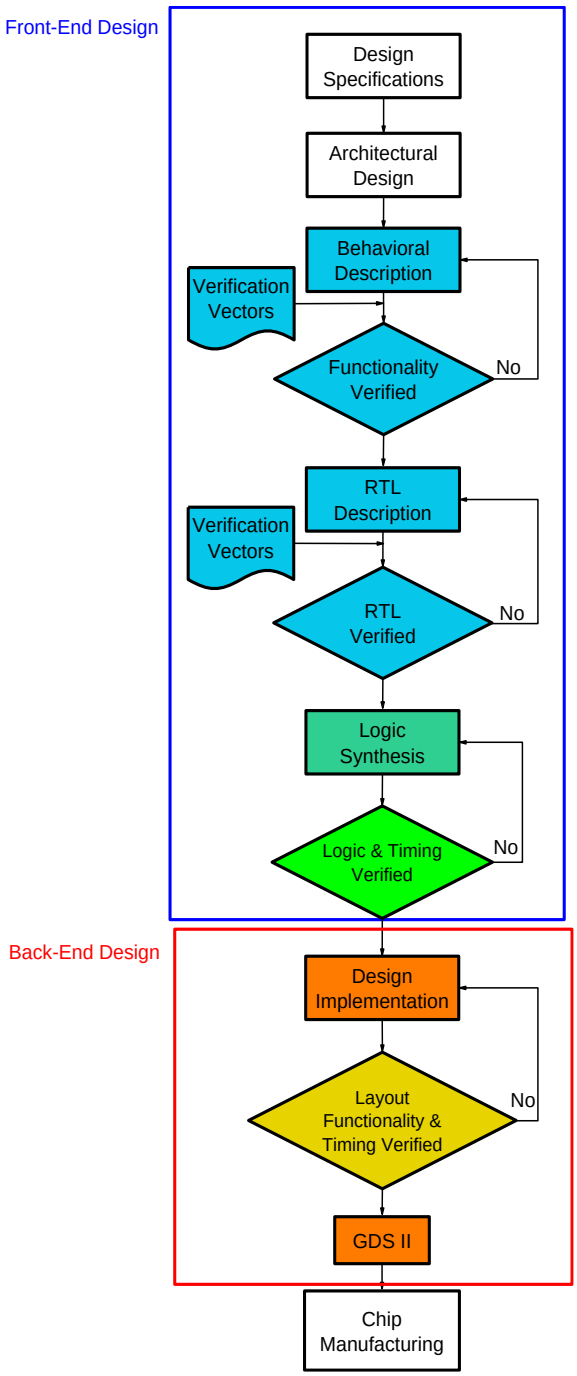

Fig. 1.1 General digital ASIC design flow.

The RTL description is a refined model describing the circuit in terms of hardware registers, combinational logic, and the data flow between them for implementing the designed architecture.

The RTL model should be verified by exhaustive pre-synthesis simulations for ensuring

that the design will meet specifications.

1.1.1.1 Logic Synthesis

The logic synthesis[4] stage [Franzon-99], is a process by which the abstract form of the designed circuit (RTL model) is transformed into a design implementation in terms of basic logic blocks taken from a standard-cell library. Standard cells are logic blocks created by every ASIC manufacturer using known functional, physical, and electrical characteristics. Therefore, they can be represented by third-party tools enabling performance of full implementation of a very high gate density and good electrical performance designs, based on standard cells without using the full factory-specific models. Steps of logic synthesis are shown in Fig. 1.2 and commented below.

The first phase of logic synthesis receives the RTL model and translates this middle abstraction level HDL file to generic combinational logic and memory elements.

In the Logic Optimization step, equations representing logic circuits are minimized, flattened and factorized.

The Logic to Technology step translates the optimized logic level description to a gate level description, using standard cells from a specific technology library. The resulting collection of standard cells, plus the required electrical interconnections is called gate-level netlist.

The Time and Area Optimization step optimizes the gate-level description, using cell substitution for meeting specific area and timing constraints.

Logic synthesis produces a gate-level netlist of the optimized circuit with accurate cell timing information. This stage ends with post-synthesis simulations for verifying that the gate level circuit fully provides the desired functionality and meets the appropriate timing requirements.

1.1.2 Back-End Design

In the second part of digital ASIC design flow, physical implementation receiving the gate-level netlist (standard cells and interconnects) converts it into geometric shape representations,

[4] U. of Colorado at Boulder, Getting Started with RTL Compiler. Feb. 02, 2014, http://ecee.colorado.edu/~ecen5837/cadence/RTL_synthesis.html.

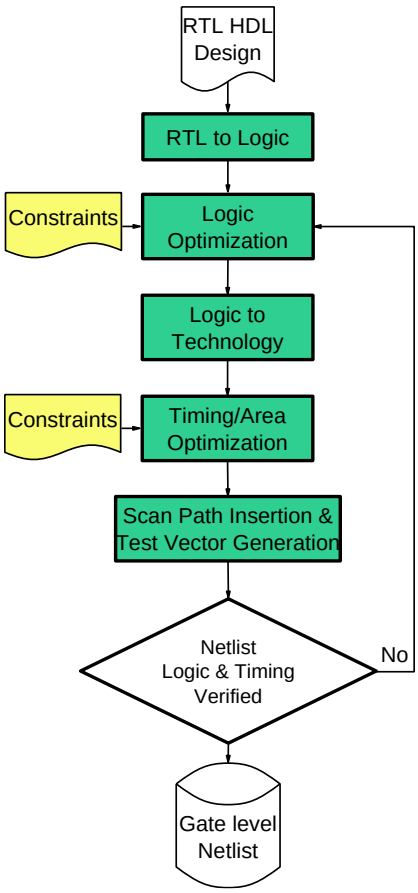

Fig. 1.2 General logic synthesis steps.

which are transformed in the corresponding layers of materials when the chip is manufactured. This geometric representation is called integrated circuit layout. Physical implementation steps, depicted in Fig. 1.3, include both design and verification of layout. This thesis document does not describe details of digital implementation flow because some Cadence tools for this purpose have already installed in the ITESO integrated circuits laboratory. Additionally, a first version of a tutorial was reported explaining, step by step, how the digital design implementation stage using CAD software of Cadence [Castorena-13] is performed. Furthermore, some tutorials introducing this stage are reported in [Farmer-11], and [Shan-08].

The manufacturing process performed at fab houses follows physical implementation stage.

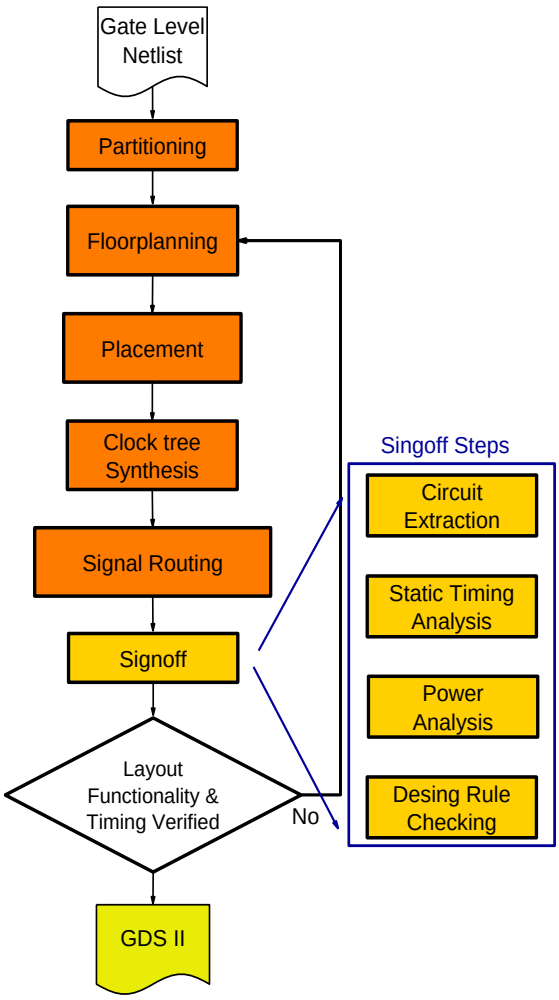

Fig. 1.3 Digital physical design implementation flow.

1.2. Cadence Tools for Digital ASIC Design Flow

There are several tutorials and web pages[5, 6]of universities dealing with ASIC design flow using CAD tools, however the information is incomplete or restricted to enrolled students [Franzon-99], [Dharwadkar-10], [Farmer-11], [Shan-08]. Some of these public tutorials are not updated or are very specific to the set of tools installed in their labs [Theocharides-05], [Engel-

[5] George Washington University, Design & Testing of VLSI Circuits. Feb. 02, 2014, http://www.seas.gwu.edu/~vlsi/ece128/SPRING/lab.html.

[6] Auburn University, Computer-Aided Design of Digital Circuits. Aug. Jan. 08, 2015, http://www.eng.auburn.edu/~nelson/courses/elec5250_6250/.

10], [Gurkaynak-06][7-11]. Many tool names are mentioned in these documents, some of which were discontinued[12] or the tool names were changed. Table 1.1 shows some names and descriptions of Cadence tool that were found in the bibliography.

This thesis aims at proposing a flow for implementing digital ASIC design using current Cadence CAD tools available at ITESO. Many tools for assisting integrated circuit design, divided into several categories[13] were found on the Cadence web page[14]. Considering only the "logic design" category, there are 16 products for supporting and verifying different logical design phases[15]. Table 1.2 shows these phases and the Cadence recommended tools.

Four categories were established for simplifying the variety of Cadence tools in order to make an initial selection for study at ITESO:

i. Logic design: the main product in this group is Encounter RTL Compiler[16], the key tool for performing top-down global RTL design synthesis on standard cells.
ii. Digital implementation: the central tools for this are Encounter Digital Implementation System (formerly known as SOC Encounter) and First Encounter.
iii. Analysis and signoff: Encounter Timing System and Encounter Power System can be used for performing some tasks at this ASIC design flow stage.
iv. Design Verification: Encounter Conformal and Incisive Formal Verifier are the recommended Cadence tools for design and verification.

[7] MIT Open Course Ware, Complex Digital Systems. Jan. 08, 2015, http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-884-complex-digital-systems-spring-2005/.
[8] MIT Open Course Ware, Communication System Design. Jan. 08, 2015, http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-973-communication-system-design-spring-2006/index.htm.
[9] Polytechnic Institute of NYU, CAD Tool Tutorials. Feb. 02, 2014, http://eeweb.poly.edu/labs/nanovlsi/tutorials.html.
[10] U. of Virginia, RTL Logic Synthesis Tutorial. Feb. 02, 2014, http://www.ee.virginia.edu/~mrs8n/soc/rc_tutorial.html.
[11] NCSU EDA Wiki, Tutorials. Feb. 02, 2014, http://www.eda.ncsu.edu/wiki/Tutorial:Contents.
[12] Cadence-Online Support, Product & Release Lifecycle. Jan. 08, 2015, http://support.cadence.com/wps/myportal/cos/COSHome/resources/lifecycle/.
[13] Cadence-Tools, Integrated Design and Verification Technologies, Methodologies, and Application Specific Kits. Feb. 02, 2014, http://www.cadence.com/products/pages/default.aspx.
[14] Cadence-Alliances, Cadence and IBM ASIC Partnership. Feb. 02, 2014, http://www.cadence.com/Alliances/asic_program/ibm/pages/default.aspx.
[15] Cadence-Tools, Logic Design. Feb. 02, 2014, http://www.cadence.com/products/ld/Pages/default.aspx.
[16] Cadence-Online Support, Encounter RTL Compiler Synthesis Flows. May. 14, 2015, http://support.cadence.com/wps/myportal/cos/!ut/p/a0/Rcc7EoMgEADQs6SwFTCOxjAUdjlCYuMssAZGRMKSz_GTLt17bGJXNkV4-TsUv0cIv98sYgo-rpuPZ7sbSdmo9NQkH6ri2cxUIJe_RFs3FcdPykg0L2F_165sQRJCNu4C5BRvBjgdte6F5nZouwVaYQQg6L6zAjlL63j4AvKpJoM!/.

TABLE 1.1. SOME DIGITAL ASIC DESIGN CADENCE TOOLS

Cadence tool name	Description / Comment	Cadence Release
NC Verilog Simulator	This tool is ideal for architecture analysis, system-level verification, and embedded software development, and it supports transaction recording and analysis for the SystemC® Verification Library. / Products Incisive NC were discontinued in October 6 2008, but in the releases INCISIV 111 - INCISIV 131 the products Incisive Enterprise simulator – XL and Cadence Simulation Analysis Environment (SimVision) are included. In some tutorials the names NC-SIM, NC-SC Simulator, NC Verilog and SimVision were found for referring to this group of tools.	INCISIV111
Cadence SimVision Debug	A unified graphical debugging environment within Incisive Enterprise Simulator supports signal-level and transaction-based flows across all IEEE-standard design, test-benches, and assertion languages, in addition to concurrent visualization of hardware, software, and analog domains. / This tool was found as Cadence Simulation Analysis Environment (SimVision)	INCISIV111
Verilog®-XL Simulator NC Verilog Simulator	Pre-Synthesis simulation, Post-Synthesis simulation. / In some tutorial this tool was found as Stand-Alone Cadence Verilog.	Included in INCISIVE 131
Encounter RTL Compiler - XL	To synthesize RTL models to standard cells using global algorithms that enables concurrent optimization of timing, area, and power intent. In some tutorials this tool was referred as Verilog-XL compiler.	RC111, RC121, RC131.
RTL Compiler Ultra	"RTL Compiler Ultra is a powerful tool for logic synthesis and analysis for digital designs. It is fully compatible with all other Cadence tools and especially with Cadence Encounter which is mainly used for physical design automation (floorplanning, placement and rooting)". / This tool was not found in Cadence page. The new version should be Encounter RTL Compiler.	
Encounter Digital Implementation System (EDI)	A powerful tool for back-end design: floorplanning, place-and-route, power and clock distribution. To generate layout from Verilog netlist. The old name for this tool was SoC Encounter RTL-to-GDSII System	EDI110-EDI132
First Encounter Design Exploration and Prototyping	For big and hierarchical designs to determine in early stages of design flow area, timing and power requirements. / In some tutorials this tool was referred as First Encounter	EDI110-EDI132
Encounter Conformal Constraint Designer	Automated validation and refinement of timing constraints	CONFRML 121 CONFRML 131
Encounter Conformal XL	To verify and debug multimillion gate designs without using test vectors. From RTL to final LVS netlist. In some tutorials this tool was referred as Encounter Conformal Equivalence Checker (EC)	CONFRML 121
Incisive Enterprise Simulator (IES)	It is a multi-language simulation, fuels testbench automation, reuse, and analysis to verify designs from the system level, through RTL, to the gate level. It supports metric driven verification, and mixed-signal verification. Is the core engine for low-power verification, working closely with Conformal LP.	INCISIV111
Encounter Timing System	Encounter Timing System is a full-chip static timing analysis (STA) solution providing gate-level delay calculation, signoff-level timing and signal integrity (SI) analysis, statistical timing and leakage analysis, advanced on-chip variation analysis, and advanced node functionality required for double-patterning and waveform effects.	ETS131

TABLE 1.2. LOGIC DESIGN PHASES AND RECOMMENDED CADENCE TOOLS

Design Task	Cadence Tools	
Chip planning	Cadence Incyte Chip Estimator Cadence Chip Planning System	
Constraint design and validation	Encounter Conformal Constraint Designer	
Logic synthesis	Encounter RTL Compiler Encounter RTL Compiler Advanced Physical Option	
Equivalence checking	Encounter Conformal Equivalence Checker	
Low power validation	Encounter Conformal Low Power	
Engineering change order	Encounter Conformal ECO Designer	
Test	Encounter Test Product Suite	Encounter DFT Architect Encounter True Time ATPG Encounter Diagnostics
Static timing analysis	Encounter Timing System	
Formal analysis	Incisive Formal Verifier	
Simulation	Cadence Low Power Methodology Kit	
Design and verification IP modeling	Incisive Verification IP	
Verification management	Incisive Design Team Manager Cadence Low Power Methodology Kit Incisive Desktop Manager Incisive Verification IP	

The following tools for ASIC front-end and back-end design are reported in [Dharwadkar-10]. Front-End design: NC Verilog® was used for functional verification, RTL description, compilation and simulation. For visualizing wave forms, SimVision® was used, for synthesis, RTL Compiler®, and for power estimation, SimVision® and RTL Compiler®.

Back-End design, also known as physical design: SOC Encounter®, currently known as Encounter Digital Implementation System® is used for partitioning, floor-planning, placement, clock tree synthesis, signal routing and timing closure.

Due to the complexity of current digital ASIC design flow, Cadence recommends learning the use of their tools gradually, step by step, and at different levels. The Fig. 1.4 shows the recommended Cadence tool learning route for digital ASIC design.

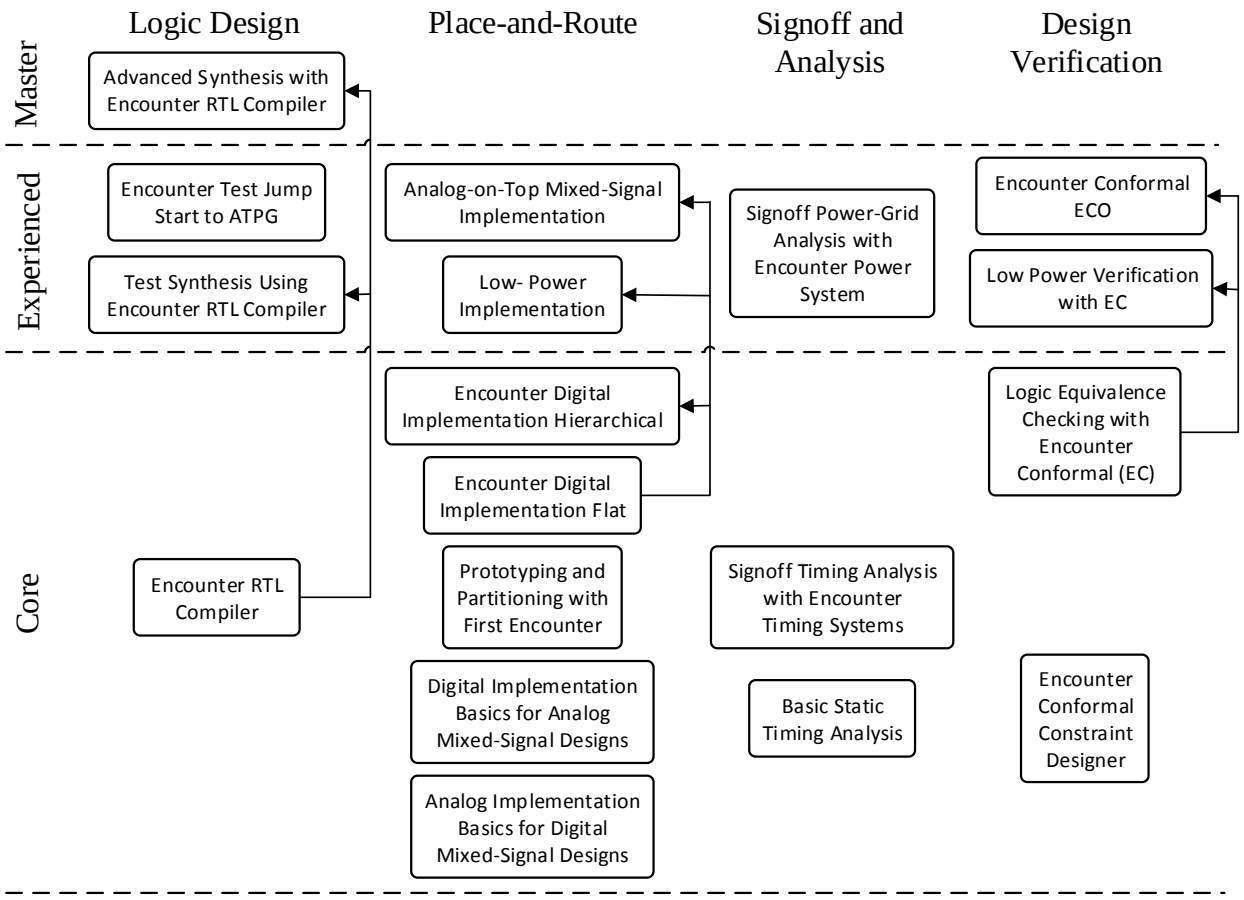

Fig. 1.4 Cadence tools training map for digital design using Encounter® technology.

Cadence has three product levels: L, XL, and GXL [Cooney-10], basic, intermediate and advanced respectively. Each tool level has different capacities, such as the number of gates in the design or the information that can be extracted from it.

After reviewing several ASIC design flow tutorials [Dharwadkar-10], [Engel-10][17-21] and examining Cadence digital ASIC design tools[22], we recommend exploring the tools reported in

[17] Virginia Tech VLSI for Telecommunications Group, VTVT ASIC Design Flow. Feb. 02, 2014, http://www.vtvt.ece.vt.edu/vlsidesign/designFlow.php.
[18] George Washington University, Design & Testing of VLSI Circuits. Feb. 02, 2014, http://www.seas.gwu.edu/~vlsi/ece128/SPRING/lab.html.
[19] Chiptalk.org, Cadence Interoperability using OpenAccess. Feb. 02, 2014, http://www.chiptalk.org/modules/wfsection/article.php?articleid=12.
[20] Polytechnic Institute of NYU, CAD Tool Tutorials. Feb. 02, 2014, http://eeweb.poly.edu/labs/nanovlsi/tutorials.html.
[21] U. of Virginia, RTL Logic Synthesis Tutorial. Feb. 02, 2014, http://www.ee.virginia.edu/~mrs8n/soc/rc_tutorial.html.
[22] Cadence-Alliances, Cadence and IBM ASIC Partnership. Feb. 02, 2014, http://www.cadence.com/Alliances/asic_program/ibm/pages/default.aspx.

TABLE 1.3. RECOMMENDED CADENCE TOOLS FOR DIGITAL ASIC DESIGN AT ITESO

Design Stage	Name Tool	Cadence Release
Behavioral modeling and verification	System C, Incisive Enterprise simulator	XL131
RTL description and simulation	Incisive Enterprise simulator	XL131
RTL synthesis	Encounter RTL Compiler	RC121
Design implementation	Encounter Digital Implementation System	EDI131
Verification, analysis and signoff	Conformal XL, Encounter Timing System	CONFRML121, ETS131

Table 1.3 for performing this kind of design at ITESO. The following paragraphs indicate the digital ASIC design stages and the recommended Cadence tools.

For behavioral modeling and verification, System C language and the Incisive Enterprise simulator XL 13.1 can be used. For putting RTL description and simulation into practice, Incisive Enterprise simulator XL 13.1 is also recommended. For accomplishing RTL synthesis, Encounter RTL Compiler 12.1 is the adequate tool[23, 24].

The digital design implementation stage can be realized using Encounter Digital Implementation System 13.1. This tool is useful for design partitioning, floorplanning, placement, clock-thee synthesis and routing. Additionally, some verification, analysis and signoff activities can be performed using Encounter Timing System 13.1 and Encounter Conformal XL 12.1. Table 1.3 shows Cadence tools recommended for performing each of these stages at ITESO and in Fig. 1.5 digital ASIC design flow using Cadence tools is depicted.

[23] Cadence, Encounter® RTL Compiler Synthesis Flows, Online Support Resources. Jan. 08, 2015, http://support.cadence.com/wps/myportal/cos/!ut/p/a0/Rcc7EoMgEADQs6SwFTCOxjAUdjlCYuMssAZGRMKSz_GTLt17bGJXNkV4-TsUv0cIv98sYgo-rpuPZ7sbSdmo9NQkH6ri2cxUIJe_RFs3FcdPykg0L2F_165sQRJCNu4C5BRvBjgdte6F5nZouwVaYQQg6L6zAjlL63j4AvKpJoM!/.

[24] Cadence-Community-Logic Design, RTL Compiler Beginner's Guides Available on Cadence Online Support. Feb. 02, 2014, http://www.cadence.com/Community/blogs/ld/archive/2013/11/12/rtl-compiler-beginner-s-guides-available-on-cadence-online-support.aspx.

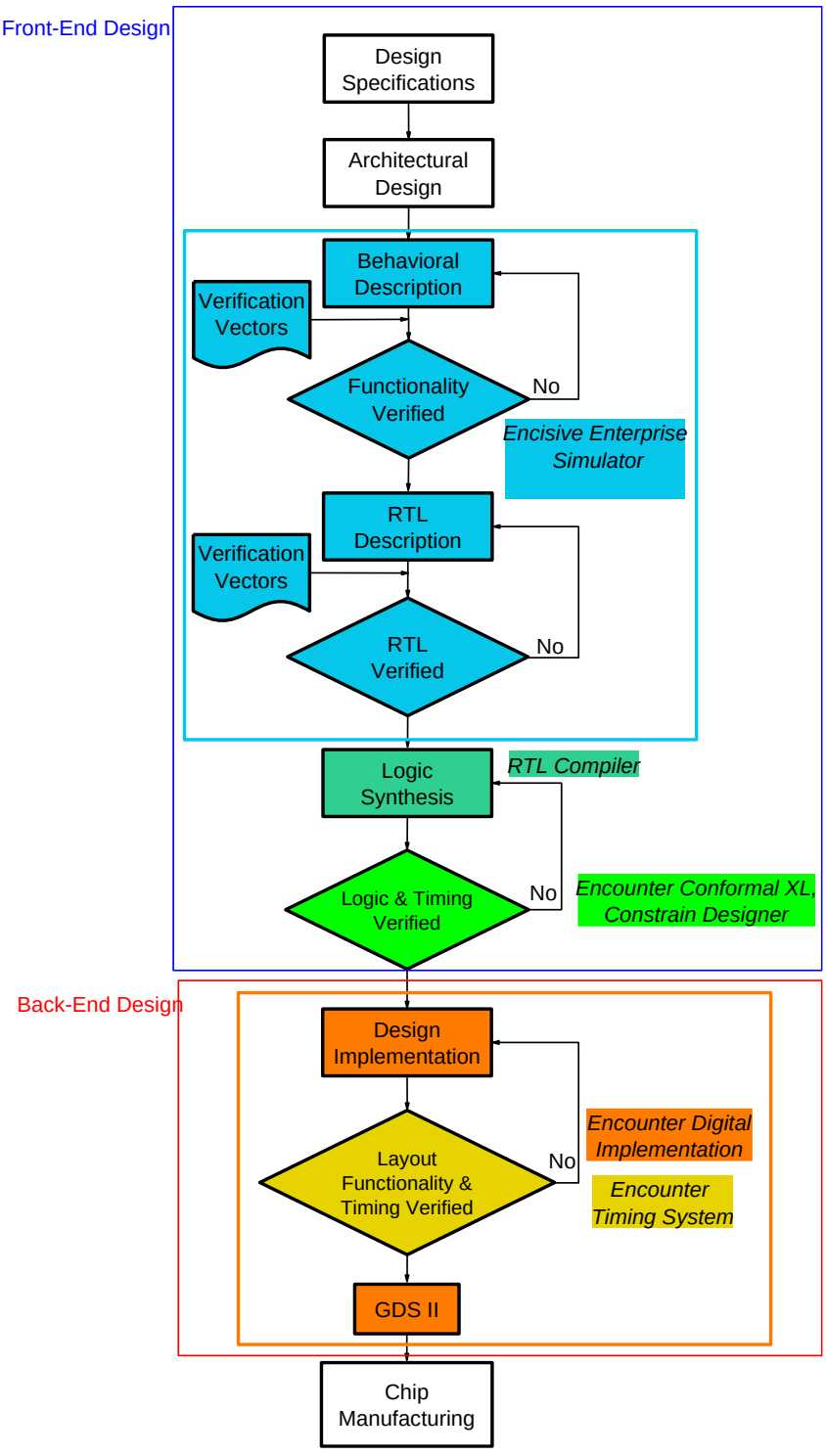

Fig. 1.5 Cadence tools for performing digital ASIC design flow.

1.3. Logic Synthesis Process

In the previous section, Encounter RTL Compiler was chosen to perform ASIC logic synthesis. An overview of the generic RTL Compiler workflow is presented below.

The basic necessary inputs to Encounter RTL Compiler (RC) for synthesizing design are:

i. Descriptions of the circuit using a hardware description language (HDL), such as Verilog or VHDL at the register transfer level (RTL) abstraction.
ii. Technology libraries for both standard cells and hard-macros.
iii. Design Constraints in either SDC or native RC format.
iv. A script file with compilation directives.
v. Physical Data (optional) such as library exchange format libraries (LEF), cap-table and design exchange format floorplan (DEF).

The HDL files must be the latest version of the RTL verification process, when the functionality of the circuit passes the test.

The standard-cell library has timing information for the specific technology to be used for implementing the circuit.

The script file should have at least the follow information: names of the RTL design files, the design directory path, and the name of the top-level module and must specify the design maximum working frequency. The synthesis tool aims at optimizing the design for meeting working frequency requirements, based on the information provided.

The recommended directory structure for a design logic synthesis using Encounter RTL Compiler tool is shown as follows [Dharwadkar-10].

```
/$USER
        /Cadence
                /Design_name_directory
    design_name_file.v
    test_bench_file.v
 /Encounter _directory
    encounter_configuration_file.conf
    encounter.tcl
    encounter_power.tcl
    gds2_encounter.map
    timing_standar_cells.v
```

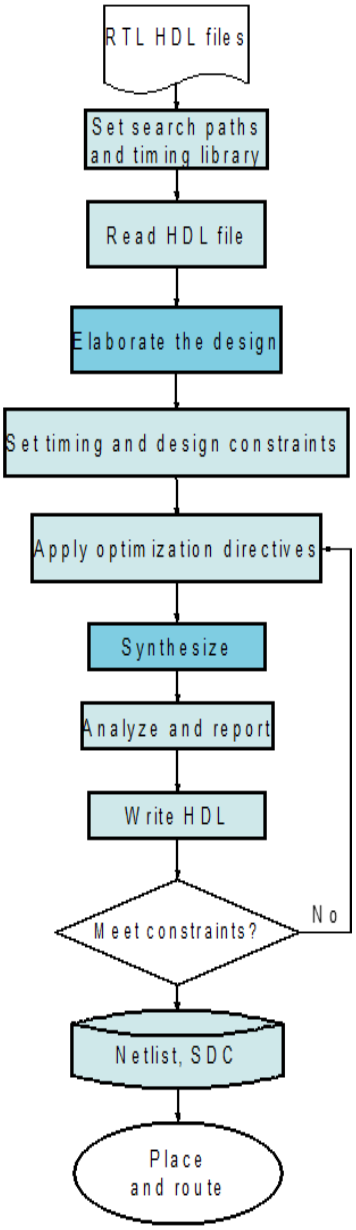

Fig. 1.6 Generic RTL Compiler® workflow.

Providing the physical data mentioned above in the point v. is optional. However, it is advised when readily available. Fig. 1.6 depicts generic RTL Compiler workflow.

The Fig. 1.6 shows the most important steps in the logic synthesis using Encounter RTL Compiler, those are fundamental in determining the overall optimization strategy, which is controlled by commands, attributes and variables by the optimization script.

1.3.1 Elaboration Step

Elaboration involves several checking and optimizing designs. The elaborate command automatically elaborates the top-level design and all of its references by propagating parameter values specified for instantiation.

In the elaboration step the RTL Compiler performs the following tasks: a) building data structures; b) inferring design registers; c) performing higher level HDL optimization, such as dead code removal; d) checking semantics. Additionally, if there are any gate-level netlist with the RTL files, RC automatically links the cells to their references in the technology library. Users need not issue any additional linking command.

After elaboration, RC has an internally created data structure for the whole design, so users can apply constraints and perform other operations. Users can generate a generic netlist for a specific Verilog module and all its sub-modules using the following command: elaborate <*top_module_name*>. For more information about commands and attributes see[25].

1.3.2 Synthesis Step

Synthesis is the process of transforming the RTL-HDL design into a gate-level netlist, given all the specified constraints and optimization settings. Within RC, synthesis is performed in the following two phases: a) synthesizing the design to generic logic (RTL and data-path optimizations are performed in this step); b) mapping the technology library and performing incremental optimization. These two sequential steps can be performed by the synthesize command options -to_generic, and -to_mapped respectively.

In the generic synthesis phase, RC performs technology-independent optimizations, including constant propagation, resource sharing, logic speculation, multiplexor optimization, and carry save arithmetic optimizations. Users can run this phase with the synthesize –to_generic -effort <*effort_level*> command. The medium effort is the default choice, but users can use high effort for data-path intensive designs, or designs for which it is hard to meet timing.

[25] Cadence-Community-Logic Design, RTL Compiler Beginner's Guides Available on Cadence Online Support. Feb. 02, 2014, http://www.cadence.com/Community/blogs/ld/archive/2013/11/12/rtl-compiler-beginner-s-guides-available-on-cadence-online-support.aspx.

In the mapping-synthesis phase, RC maps the generic gate-level netlist to the technology library cells. For performing this phase, user can run the synthesize –to_map -no_incremental -effort <*effort_level*> command.

The incremental synthesis (IOPT) is the final optimization phase in the synthesis process. The primary intent of this stage is cleaning up timing by using local optimizations, such as critical region synthesis (CRR), and inserting scan chains if it is enabled. All constraint violations arising from max_cap, max_trans, and max_fanout, are considered and subsequently fixed in this step. Optimizations performed during IOPT synthesis improve timing and area and fix design rule checking (DRC) violations.

Timing has the highest priority by default, and RTL Compiler will not fix DRC violations if it causes timing violations. This priority can be overridden by setting the drc_first attribute to true. In this case, all violations will be fixed as well as those paths with positive slack. Optimizations performed during this phase include multi-bit cell mapping, incremental clock gating and retiming, tie cell insertion and assign removal[26].

Users can run this stage using the synthesize –to_map -incremental -effort <*effort_level*> command.

1.3.3 Analysis and Reports

Encounter RTL Compiler can generate reports which allow analyzing the synthesis results. The report timing command should be used for generating reports on the timing of the current design. The default timing report generates the detailed view of the most critical path in the current design. The timing report provides the following information: a) type of cell (or, nor, and gates, flip-flop, etc.); b) the cell's fan-out and timing characteristics (load, slew, and total cell delay); c) arrival time for each point on the most critical path.

Use the -from and -to options for reporting the timing value between two points in the design. The timing points in the report are designated by the "<<<" indicator.

Encounter RTL Compiler also can generate a detailed area report giving the area of each

[26] Cadence-Community-Logic Design, RTL Compiler Beginner's Guides Available on Cadence Online Support. Feb. 02, 2014, http://www.cadence.com/Community/blogs/ld/archive/2013/11/12/rtl-compiler-beginner-s-guides-available-on-cadence-online-support.aspx.

component in the current design, based on the specified technology library. The report gates is the command for generating a report of the gate profile being used in the design.

An example of a simple script delineating the very basic Encounter RTL Compiler flow is shown below:

```
set_attribute lib_search_path <full_path_of_technology_library_directory> /
set_attribute hdl_search_path <full_path_of_hdl_files_directory> /
set_attribute library <technology_library> /
read_hdl <hdl_file_names>
elaborate <top_level_design_name>
read_sdc <sdc_file_name>
set clock [define_clock -period <periodicity> -name <clock_name> [clock_ports]]
external_delay -input <specify_input_external_delay_on_clock>
external_delay -output <specify_output_external_delay_on_clock>
synthesize -to_mapped
report timing > <specify_timing_report_file_name>
report area > <specify_area_report_file_name>
write_hdl > <specify_netlist_name>
write_script > <script_file_name>
```

Additionally, the write_template command can be used and subsequently modifying a basic script skeleton and then modify the same to suit specific designs. These modifications are minimal and usually contain adding design inputs such as libraries, RTL files and constraints.

Complete and detailed information on the Encounter RTL Compiler workflow and command for performing the synthesis process can be found in [Cadence-14].

1.4. Conclusions

In this chapter, the general digital ASIC design flow was described, and some Cadence tools were presented, a specific selection of which was proposed to be installed at ITESO integrated circuit laboratory for digital ASIC designing.

The logic synthesis steps were presented in detail given that the tools required for performing this stage have not yet been installed in the ITESO integrated circuits laboratory and it is of first importance for having the capability to perform full digital ASIC design flow.

This chapter compiles information sources which will be useful to implement the digital ASIC design flow at ITESO integrated circuit laboratory. This will enable professors to implement

and teach digital ASIC design in graduate courses, as well as to support research projects.

2. Implementation of a Digital VLSI Front-End Design at ITESO

In this chapter, the fundamental components for implementing digital VLSI front-end design at ITESO are introduced. Some process design kit (PDK) concepts are presented and the necessary libraries and technologies enabling ITESO students and teachers to do digital VLSI designs are identified. MOSIS offers several integrated-circuit fabrication technologies; the most appropriate one is selected in order to students and teachers are able to do digital VLSI designs and to be manufactured by this company.

The concepts and characteristics of the North Carolina State University Cadence Design Kit (NCSU CDK) and the Virginia Tech VLSI for Telecommunications (VTVT) standard-cell library are presented. These are basic for understanding and performing automated digital VLSI circuit design. The NCSU CDK and the VTVT standard-cell library are selected to implement for the first time a digital VLSI front-end design in the ITESO integrated circuit design laboratory.

Furthermore, this chapter describes the details of how to perform the logic synthesis of a basic sequential digital circuit following the RTL Compiler synthesis flow. The best way to run RC and controlling the logic synthesis results is through a tool command language (TCL) script file, which has the attribute definitions, commands, and compilation directives. This chapter explains the elaboration and contents of an RC synthesis TCL script file for controlling and administering the task execution of a Cadence recommended synthesis flow, which is implemented and verified by synthesizing a frequency divider circuit. The inputs to the RC logic synthesis process and the synthesis results of the design example are discussed.

2.1. Selection of Integrated Circuit Fabrication Process

The most simple and economic way to access the fabrication of an integrated circuit for teaching and academic research projects is through MOSIS enterprise. ITESO has an agreement with that company, but a selection of the fabrication technology of integrated circuits is necessary in order to implement the complete digital VLSI design flow based on standard cells. We need the

PDK information and the compatible standard-cell library that allow the automation of digital integrated circuit design. Both components are described in this section.

With the purpose of making the best selection of the technology to usage at ITESO, in the following section, we will describe the MOSIS options for designing and fabrication digital VLSI integrated circuits. The defined criteria in order to choose the best technology for ITESO academic projects are:

i. Availability of the PDK.
ii. Availability of the standard-cell library compatible with the selected PDK.
iii. PDK compatibility with Cadence tools version installed in the ITESO integrated circuit laboratory.
iv. The selected technology (feature size) must be of interest for research publication, such that it facilitates scientific publications in renowned journals.

The fabrication cost should be affordable for ITESO.

2.1.1 MOSIS Fabrication Processes

MOSIS offers multiples integrated circuits fabrication technologies from six different companies. To access these ones, MOSIS has three kinds of accounts for academic and research institutions. Each kind of account defines the PDK to which the institution is able to access. Table 2.1 shows a summary of MOSIS integrated circuit fabrication processes. It is shown the IC manufacturer, the feature size, the MOSIS account kind required to access the PDK, and the standard cells availability in order to do automated digital VLSI design.

2.1.2 Kinds of MOSIS Academic Accounts and Their Characteristics

For academic and research institutions, MOSIS offers three kinds of accounts to access its IC fabrication services: Instructional, Research, and Commercial[27]. Each account kind has different technology access restrictions: maximum silicon area, number of chips per year, price, etc. For example, the MOSIS Educational Program (MEP) Instructional account has a quantity of

[27] The MOSIS Service, MOSIS FAQs: MOSIS Educational Program (MEP). Jun. 05, 2014, http://www.mosis.com/pages/Faqs/faq-education#16.

TABLE 2.1. SUMMARY OF MOSIS INTEGRATED CIRCUIT FABRICATION PROCESSES

IC Manufacturer	Feature Size (nm)	MOSIS Account Kind	Available Standard Cell
TSMC	90, 65, 45, 40	Commercial	ARM
TSMC	TinyChip (90, 65)	Commercial	ARM
Global Foundries	350, 180, 130, 65, 28 CMOS	Commercial	-
IBM	130 (8XP), 180 (7WL) BiCMOS	Commercial	-
IBM	180 (7RF SOI)	Commercial	-
IBM	130 (8HP) BiCMOS	Commercial, Research	Limited
IBM	180 (7RF CMOS)	Instructional	-
IBM	130 CMOS (8RF-DM)	Commercial, Research	ARM
ON Semiconductor	500 CMOS (C5N)	Instructional, Research	-
ON Semiconductor	700, 500, 350 CMOS	Commercial	-
AMS	350 CMOS, HVCMOS, BiCMOS	Commercial	-
AMS	180 CMOS, HVCMOS	Commercial	-
Imec- ePIXfab	SiPhotonics	Commercial	-

allocations which are expressed in TinyChip units, rather than dollars or number of integrated circuits. A TinyChip unit for a given process represents both allocation of project area and a quantity to be delivered.

MEP Instructional designs must fit into MOSIS TinyChip units. To fit into one TinyChip unit, a project designed in either the ON Semiconductor 0.50 micron (C5) or IBM 0.18 micron (7RF) technology must be no larger than 1.5 mm×1.5 mm. MEP Instructional designs may be larger, by using more TinyChip units in multiples of 1.5 mm×1.5 mm, up to 3.0 mm×3.0 mm which would be 4 TinyChip units.

Table 2.2 shows a summary of MOSIS accounts, their characteristics, and restrictions.

From Table 2.1 and Table 2.2 we emphasize that the technologies which can be accessed

TABLE 2.2. MOSIS EDUCATIONAL PROGRAM ACCOUNTS AND THEIR CHARACTERISTICS

Characteristic	Account		
	Instructional	Research	Commercial
Processes Available	On Semi 0.5 µm CMOS (C5N) IBM 180 nm CMOS (7RF)	On Semi 0.5 µm CMOS (C5N) IBM 130 nm BiCMOS (8HP) IBM 130nm CMOS (8RF-DM)	All available
Area Max (Si)	3 mm × 3 mm	16 mm^2	Unlimited
Number of Designs	According to authorized budget (TinyChip)	1 project/university/year	Unlimited
Price	Free	Cost of packaging	Quotation required
Use	Projects designed by students	Unfunded research projects, thesis works, paper, journal article	Academic
License time	October 1 through September 30	Not expire	Not expire
Chips per design	On C5: 5 IBM 7RF: 40	40	According to order
Packaging restrictions	Ceramic or Open Cavity Plastic (OCP)	According to order	According to order

with the MOSIS Instructional account do not have complete access to standard-cell libraries, so automation of digital VLSI design cannot be performed using an Instructional account.

The MOSIS Research account is attractive because it allows access to the On Semiconductor 0.5 µm technology and the IBM 130 nm BiCMOS and CMOS technologies. For the last one, there is standard-cell library[28] from ARM[29]. This means that we can do automated digital VLSI designs using the IBM 8RF-DM technology and manufacture these by covering the packaging cost[1] only. The disadvantages of this type of account are that only one design per year per university can be manufactured, and the access to the kit must be requested through a MOSIS

[28] The MOSIS Service, IBM Design Rules and Cell Libraries. Jun. 16, 2014, http://www.mosis.com/vendors/view/ibm/documents.

[29] The MOSIS Service, ARM Library Access for Universities. Jun. 05, 2014, http://www.mosis.com/pages/Technical/Designsupport/artisan-university.

project application.

2.2. Fundamental Components for Automated Digital VLSI Design

Three fundamental components are required for implementing digital VLSI design flow: EDA tools, a process design kit, which has the information about the technology of the integrated circuits fabrication process, and libraries with timing and physical data of the basic building blocks named standard cells. Virginia Tech University developed a standard-cell library which is available for academic and research projects at universities[30]. This library was designed using the NCSU CDK installed in the ITESO integrated circuit laboratory. The VTVT standard-cell library was installed as part of this project. Some concepts and characteristics of these two components are described below.

2.2.1 The Process Design Kit of North Carolina State University

"The North Carolina State University Cadence Design KIT (CDK) is a collection of technology files, custom SKILL routines, parts libraries, and Diva rules files aimed at facilitating full-custom CMOS IC design through MOSIS. The CDK is used at N.C. State University in both teaching and research, and it has been used to fabricate working chips" [Schaffer-98].

"The NCSU CDK focuses on providing the means to do full-custom CMOS IC design (SCMOS design rules) through MOSIS, including schematic entry, Verilog digital simulation, analog circuit simulation, layout DRC checking and device extraction, and mask generation"[31].

Some of the NCSU CDK features are:

i. Provides interface to HSPICE/Spectre through Analog Artist, with MOSIS-provided transistor models in place, as well as with interface to Verilog with technology-independent parts.

ii. Technology-independent libraries for analog and digital parts. These parts have SKILL

[30] Virginia Tech VLSI for Telecommunications, Cell Libraries to Support VLSI Research and Education. Jul. Mar. 31, 2014, http://www.vtvt.ece.vt.edu/vlsidesign/cell.php.
[31] NCSU EDA Wiki, NCSU CDK overview. Jun. 10, 2014, http://www.eda.ncsu.edu/wiki/NCSU_CDK_overview.

code hooked in to enforce sizing and grid rules, automatic transistor model selection depending on technology, and drain/source area/perimeter estimation.

iii. Technology libraries, one library for every MOSIS SCMOS process with parameterized layout cells setup for both manual use and layout synthesis via Virtuoso-XL.

iv. MOSIS wirebond pads (HP 0.6 µm; AMI 0.6 µm; TSMC 0.40 µm).

v. Various user-friendly GUI enhancements: a) simplified library creation and technology file attachment for MOSIS technologies; b) click on any object to print info about it in the Command Interpreter Window (CIW); c) enhanced label creation (Virtuoso); d) align layout objects (Virtuoso); e) Perl/Tk program (BitGen) to easily convert 1's and 0's into analog voltage sources suitable for circuit simulation in programs as SPICE and Spectre; f) create a "publication-quality" schematic from a working schematic.

vi. Documentation of all customizations in HTML.

Some things that the NCSU CDK does not have or cannot do are: a) provide a standard-cell layout library; b) physical implementation (place-and-route stage); c) digital timing analysis; d) parasitic resistance extraction.

In the ITESO integrated circuit laboratory, the currently installed version of the NCSU CDK is the 1.6.0 beta. This kit is not yet fully supported but a related technical forum for discussing problems and solutions is available[32].

2.2.2 Standard-Cell Libraries

Automated digital VLSI circuit design is currently based on standard-cell libraries and synthesizers. This approach has the following benefits: each cell is full custom designed, supports logic synthesis, automatic layout generation, system physical design and testing; all of these with the assistance of CAD tools.

A standard-cell library is a set of basic logic blocks, for example, gates, adders, buffers, and flip-flops, which are used as building blocks of digital ASICs. Each block or cell is implemented at a physical level. Its full-custom layout is optimized to minimize the required silicon area and delays. Standard cells are characterized by having a fixed height, which allows

[32] Chiptalk.org, Forum. Jun. 23, 2014, http://www.chiptalk.org/modules/newbb/index.php.

them to be arranged in rows to facilitate the automatic generation of digital ASIC layout.

The main components of a typical standard-cell library are:

i. View database.

Consist of different level abstraction models named views, which are needed to support the digital VLSI circuit design flow:

a. Logical view of a standard cell: The cell's Boolean function whose behavior can be captured by a truth table or Boolean algebra equation using a hardware description language.
b. Schematic view: a graphical view of the transistor design netlist.
c. Layout view: is the physical representation of the cell, consisting of several layers, which correspond to different structures of transistor devices, interconnect wiring layers, and via layers, which join together the terminals of the transistor and circuit formations.
d. Abstract view: is similar to the layout view but contains much less information than the layout and may be recognizable as a Library Exchange Format (LEF) file or an equivalent. The abstract view is useful for place-and-route tools in the digital implementation stage.

   The abstract view provides information like [Patel-08a]: cell name; site name and cell orientation; cell PNR boundary; pin names, locations, pin metal layer, type and direction (input/output/input-output); location of all metal track and vias in the layout (obstructions).

   The LEF file contains technology information along with all the cell description, use an ASCCI data format to describe the standard-cell structure. Incudes the design rules for routing, and the abstract view. A LEF file contains the following sections: a) technology: layer, design rules, via definitions, metal capacitance; b) site: site extension; c) macros: cell descriptions, cell dimensions, layout of pins and blockages, and capacitances.

   To know about the structure and syntaxes of LEF file read [Patel-08b].

ii. Timing abstract file (.lib).

Provides functional definition, timing, power consumption, and noise information for each cell. Generally, in liberty format (.lib), which is an ASCII representation of the timing and power

parameters associated with any cell in a particular semiconductor technology, these parameters are obtained by simulating the cells under a variety of conditions and the data is represented in the .lib format.

The .lib file contains timing models and data to calculate: I/O delay paths, timing check values, interconnect delays. More information of liberty file can be found in [Patel-08c].

As we can see from previous sections, a standard-cell library contains different views, and information formats for each cell. Several tools are needed to use or design them. Table 2.3 shows the formats and tools to design a standard cell using Cadence tools [Patel-08d].

Access to several standard-cell design tutorials can be found at the University of New Mexico[33].

2.2.3 Virginia Tech Standard-Cell Library

IC manufacturers impose access restrictions on their standard-cell libraries because they are considered intellectual and technological properties of great economic value. These restrictions make difficult teaching and research activities in the field of digital VLSI design. One alternative for educational institutions was proposed by the Virginia Tech VLSI for Telecommunications Group, whom developed free access standard-cell libraries for academics.

"The VTVT Group has developed three standard-cell libraries targeting the TSMC 0.18um, TSMC 0.25um, and TSMC 0.35um CMOS processes available via MOSIS. The libraries can be used with Synopsys synthesis tools and the Cadence SOC Encounter, Place/Route tool. All of the cells can be viewed and edited using the Cadence Virtuoso layout editor. The standard-cell libraries require NCSU design kit or other kits that follow MOSIS design rules. Since MOSIS DEEP design rules are used for our cell library, the NCSU design kit has been modified slightly. Changes to the NCSU kit are included in this distribution"[34].

The development of the VTVT standard-cell library is described in [Sulistyo-10].

The standard-cell library that we obtained from Virginia Tech University was in an old Cadence data base format (.cdb). To use this in the ITESO integrated circuit laboratory, the VTVT

[33] University of New Mexico, Advanced VLSI Design (ECE 595). Apr. 25, 2014, http://www.ece.unm.edu/~jimp/vlsi_synthesis/.

[34] Virginia Polytechnic Institute and State University-MICS Group, Cell Libraries to Support VLSI Research and Education. Jun. 20, 2014, http://www.mics.ece.vt.edu/ICDesign/Cell_Libraries/Overview/index.html.

TABLE 2.3. VIEWS, FORMATS, AND CADENCE TOOLS IN A TYPICAL STANDARD-CELL LIBRARY

Views	Format	Cadence Tools	Comments
Physical layout	GDS-II	Virtuoso Layout Editor, ICFB	Should follow specific design standard: Constant height, offset, etc.
Logical	Verilog, TLF, LIB	Text editor	Verilog model is required for dynamic simulation, this file should preferably support back annotation of timing information. Place-and-route tools usually can use TLF
Abstract	LEF, Milkyway (Synopsys)	Cadence Abstract Generator, place-and-route tools	LEF (Layout Extraction Format): contains information about each cell as well as technology information
Timing, power, parasitic	TLF, LIB	Spectre, RTL Compiler	Detailed timing and power simulations are performed on the Spice netlist, the results are recorded in this file, including process, temperature and supply voltage variations. Also, this file contains logical information for each cell

library was converted to the Open Access (.oa) data base in order to make this compatible with new Cadence V6 environment[35] [Brunvand-13].

In order to implement the initial digital VLSI design flow in the ITESO integrated circuit laboratory, the TSMC 0.18 µm standard-cell library was installed in the mentioned lab, with the following features:

i. 83 primitive cell layouts
ii. Synopsys synthesis (.db/.sdb) and VHDL simulation libraries.
iii. LEF file for the PNR tool.
iv. Symbols and schematic libraries of standard cells.
v. Readme files and a documentation for modification of the NCSU kit.
vi. Other documentations, including the place-and-route flow used to test the library.

After the installation, by using the Cadence Virtuoso Library Manager we can see four VTVT standard-cell views: layout, physconfig, schematic, and symbol.

In the next section, the logic-synthesis workflow using RTL Compiler Cadence tool and

[35] Washington University in Sta. Louis-EDA Wiki, How to Convert a CDB Library to an OA Library. Jun. 27, 2014, http://eda.engineering.wustl.edu/wiki/index.php/How_to_convert_a_CDB_library_to_an_OA_library.

VTVT standard-cell library is presented, it is implemented and verified by synthesizing a sequential digital circuit.

2.3. Logic Synthesis Using RTL Compiler

Logic synthesis is the process of translating a behavioral hardware description language (HDL) circuit model into another HDL model that represents the same circuit through specifying fundamental logic blocks and their interconnections between them. Therefore, the structure that is described by the new model represents the same original circuit behavior. The new model is known as structural model because it is based on logic blocks from a standard-cell library, which contains the behavioral and physical models (layouts) of the basic logic blocks [Brunvand-10].

The logic synthesis flow comprises the following steps: a) conversion of the RTL model into Boolean functions, b) technology-independent optimizations, c) technology mapping, d) technology-dependent optimizations, and e) test logic insertion [Wang-09].

The first step in the logic synthesis converts behavioral or RTL descriptions into implementations in terms of generic logic gates (AND, OR, NOT, Flip- Flops, etc.), which are not linked to any technology. This means that later, it can be selected a specific standard-cell library for the technology mapping step.

In the second step are performed technology-independent optimizations using logic function reduction methods such as Quine-McCluskey or multilevel logic optimization, being the later more suitable for standard-cell based designs [Wang-09].

In the technology mapping step, the synthesizer performs an implementation of the technology-independent optimized design using a specific standard-cell technology.

After technology mapping has been done, additional optimizations are performed such as those for timing and power consumption [Cadence-12a].

Finally, additional test logic can be inserted in the circuit to support design for testability (DFT) features.

In the next sections, the RC synthesis flow is presented through an example, and the basic commands and attributes of a TCL[36] synthesis script, required for controlling and administering

[36] Tcl Tutor, Tcl Tutor Overview. Jun. 14, 2014, http://www.msen.com/~clif/TclTutor.html.

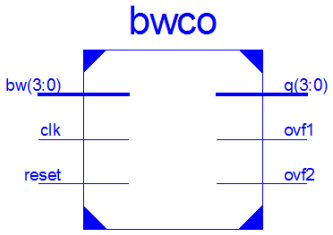

Fig. 2.1 Black-box of frequency divider circuit. Sequential digital circuit selected for implementing the Encounter RTL Compiler workflow using the Virginia Tech standard-cell library at ITESO integrated circuit laboratory.

the RC synthesis tasks, are explained.

2.3.1 Digital Circuit to Be Synthesized

The circuit example selected to implement and verify the RC synthesis flow [Cadence-14] at ITESO integrated circuit laboratory, is a basic circuit that has both combinational and sequential elements. The circuit is selected not with the goal to show the synthesis tool capacities but to practice the steps into the RC workflow, to know the RC synthesis inputs and outputs, to learn some RC fundamental commands, and for the first time, to carry out the RC synthesis workflow at the ITESO integrated circuit laboratory.

The circuit example is a frequency divider controlled by a four-bit binary word. It consists of a module-16 binary counter and a four-bit comparator-counter, the later controls the overflow time of the module-16 counter. The circuit example can generate sixteen pulse frequencies according to a four-bit input word. Fig. 2.1 shows the frequency divider black-box (bwco); it has three input ports and three output ports; whose functions are as follows:

bw (3:0): is a four-bit input port to define the output frequency on ovf2.

clk: is the clock input port for synchronizing the digital system example.

reset: is an input signal to stablish initial conditions in the system.

q (3:0): is the module-16 counter output. Its count value is incremented each ovf1 is set.

ovf1: is the comparator-counter overflow output. It is set when its count is equal to bw.

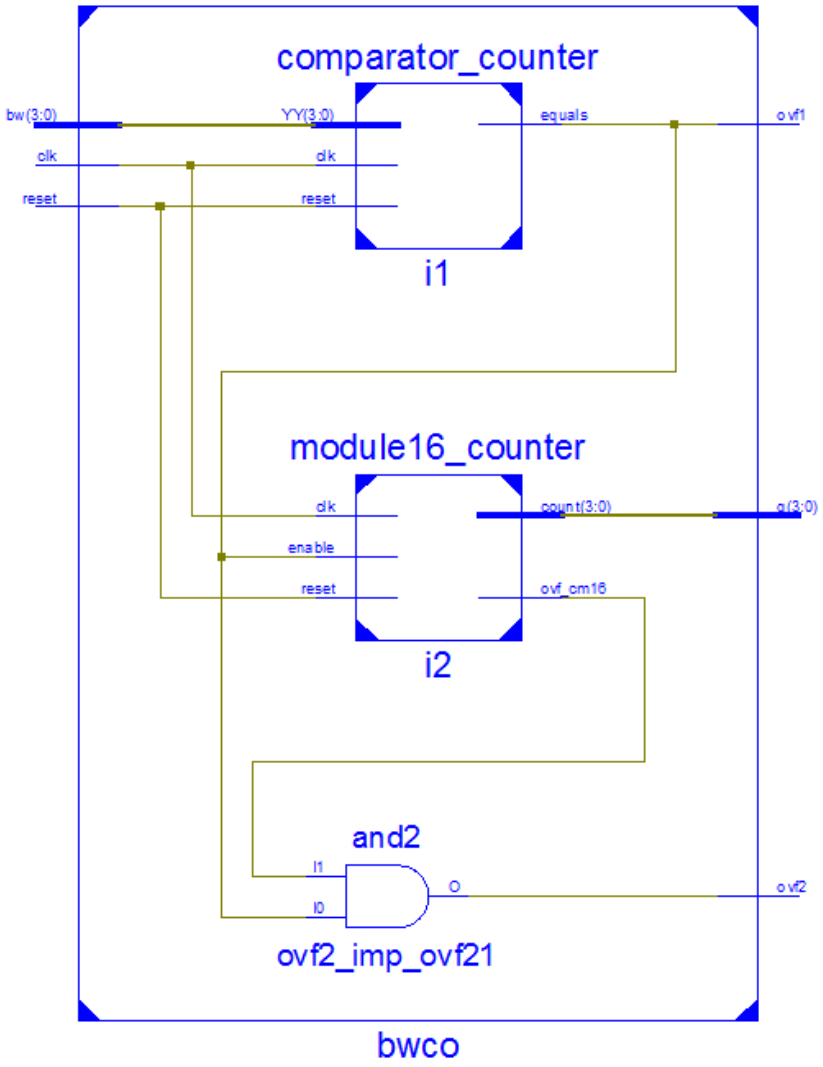

Fig. 2.2 Frequency divider block-diagram. Sequential digital circuit selected for implementing the Encounter RTL Compiler workflow using the Virginia Tech standard cell at ITESO integrated circuit laboratory.

ovf2: is the system overflow output. It is set when ovf1 and module-16 counter overflow are equals to one.

Fig. 2.2 shows the frequency divider block diagram. It has two modules: the module16_counter and the comparator_counter. Working together, these modules can generate 16 different frequencies according to clk and bw inputs. Fig. 2.3 shows simulation waveforms of the frequency divider to be synthesized.

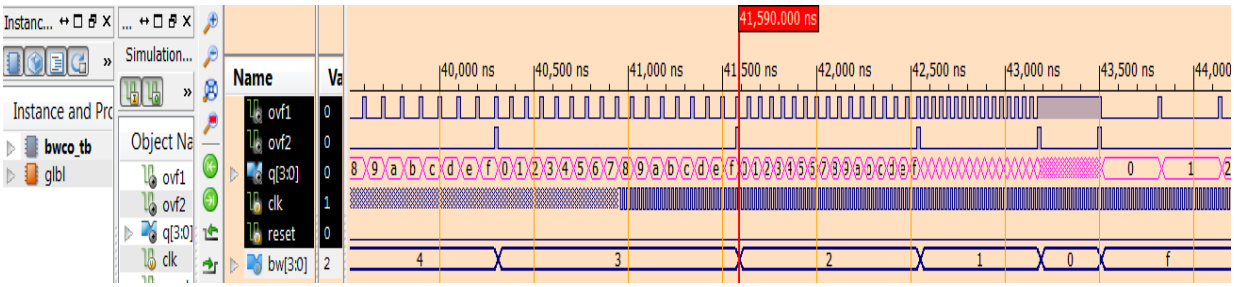

Fig. 2.3 Frequency divider circuit simulation results.

2.3.2 RTL Compiler Synthesis Flow Inputs and Outputs

Fig. 2.4 depicts a generic RC workflow. It shows the most important steps for the logic synthesis using Encounter RTL Compiler as well the inputs to each task into the synthesis process and possible outputs that the RC synthesis tool can generate.

The inputs to the RTL Compiler synthesis process are the following [Cadence-12b]:

a) The design HDL models: Verilog, VHDL, or System Verilog files.

b) The design constraints: restrictions and limitations imposed on the design, regarding the area, time, power consumption, etc. These are contained in a file that uses the Synopsys design constraints (.SDC) format or an RC native format.

c) Liberty format library: it is a file with information about the functional definition, timing, power consumption, and noise for each cell. This file uses generally the liberty (.LIB) format.

d) Library exchange format file (.LEF): is a file containing the standard-cell technology information such as layers, design rules, via definitions, metal capacitance, cell dimensions, pin layout, etc.

e) Synthesis script file (.TCL): is the file that defines variables and commands to control and administers the RC task execution of the synthesis flow.

f) Technology independent synthesizable macro-cells or IP modules required for design (GTech and Design Ware (DW) in Synopsys).

g) Optional files: capacitance table file (.CAPTBL), floorplan design exchange format file (.DEF), switching activity files (.SAIF), toggle count format (.TCF or .VCD), and common

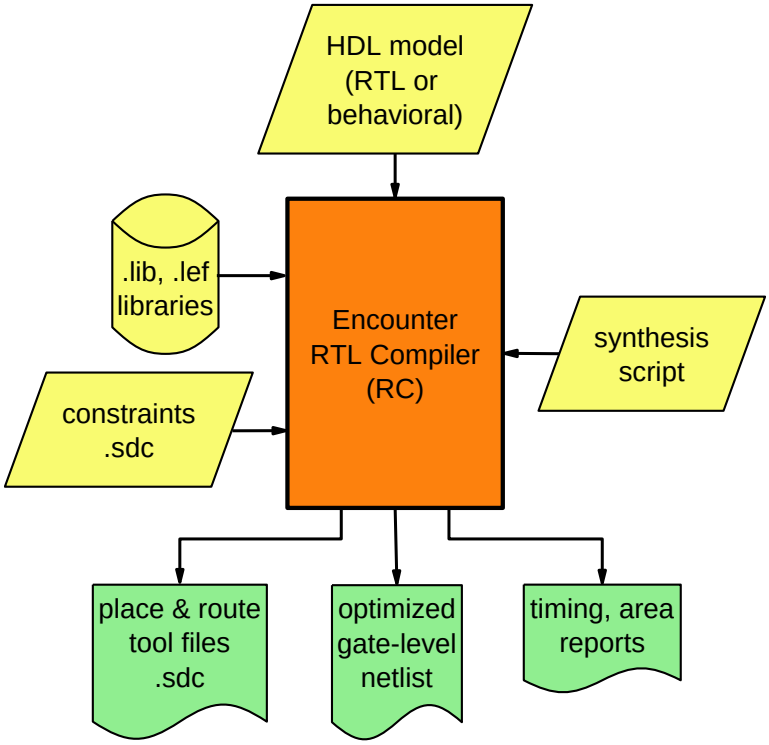

Fig. 2.4 Generic RTL Compiler synthesis workflow, inputs and outputs.

power format (.CPF) file.

The outputs from the RTL Compiler synthesis process are:

a) Optimized gate-level netlist file (.V): this file is the main product of the logic synthesis process. It is a Verilog model that describes the circuit structure based on standard cells and its connectivity, which reproduces the same functionality than the behavioral or RTL model.

b) Constraints file (.SDC): this is the current design constraint output file in Synopsys design constraint format, which is required in the place-and-route tool.

c) Optional files: scandef, DEF, Do-files, and CONF files which are useful for the back-end design stage.

The next section explains a synthesis script to control and administer a basic Encounter RTL Compiler workflow.

2.3.3 Synthesis Script for RTL Compiler

The best way to control the RC synthesis process is by using a TCL script. It is highly recommended to generate this script using the rc_write_template command. This command should be used for each new design and tool release as it ensures that all attribute and variable settings are for the latest release and guarantees the use of latest recommended synthesis flow. For the example here presented, the utilized syntax to generate a synthesis script is [Cadence-15a]:

rc_write_template –outfile *synthesis_script_bwco_sdc.tcl*

where *synthesis_script_bwco_sdc.tcl* is the TCL script file name for the design example considered. The generated script template is modified for the specific design example presented in this chapter.

To launch RTL Compiler and to execute the synthesis script, use the following command:

rc -gui -f *synthesis_script_bwco_sdc.tcl* –log *run_log_bwco*.log

where *run_log_bwco*.log is the synthesis output log file.

The next subthemes explain the sections of the RC synthesis script, *synthesis_script_bwco_sdc.tcl*, which are identified with letters from A to L in Appendix D.

2.3.3.1 Presetting Global Variables and Attributes

The Section A of the *synthesis_script_bwco_sdc.tcl* file, defines the global variables with the purpose to concentrate all the design object names in a section of the script in order to facilitate changes and reuse of the script for new designs. For example, it is useful to define global variables for the top-level design name, standard-cell libraries, and folder names for output reports.

2.3.3.2 Specifying Explicit Search Paths

The default RTL Compiler search path is the directory path where RC is launched. Specific search paths can be defined for libraries, script files, and design files. To set specific search paths, type the following set_attribute commands in the RC synthesis script file [Cadence-15b]:

set_attribute lib_search_path <*lib_path*> /

set_attribute script_search_path <*script_path*> /

set_attribute hdl_search_path <*hdl_path*> /

where *xxx_path* is the full path of the technology libraries (.LIB), script, and design files.

The slash (/) in these commands refers to the root-level RTL Compiler object containing all global RC settings. See Appendix D for specific examples of these and other commands explained in this chapter.

2.3.3.3 Setting the Target Technology Library

After setting the library search path, specify the target technology library for synthesis using the library attribute [Cadence-15b]:

set_attribute library <*lib_name.lbr*> /

After this command, RTL Compiler will use the library named *lib_name.lbr* to synthesis the design. RTL Compiler can also accommodate the .lib library format. In either case, ensure that you specify the library at the root-level (/). If the *lib_name.lbr* is not in a previously specified search path (*lib_path*), specify the full path with the *lib_name.lbr* attribute.

2.3.3.4 Setting the Synthesis Mode

RTL Compiler has two modes for synthesizing the design. The synthesis mode is determined by setting the interconnect_mode attribute, establishing whether RTL Compiler uses wire load (wireload) models or physical layout estimators (ple) during synthesis: a) wireload (default) indicates the use of wire load models for driving synthesis; b) ple indicates the use of physical layout estimators (PLE) for driving synthesis.

PLE uses physical information of the technology, such as LEF libraries and capacitance table file during synthesis, instead of the wireload model from technology library. Thus, the cell area defined in the LEF file will be used in place of those in the timing library area to provide better closure with back-end design tools. The timing library area will be used if the physical libraries do not contain any cell definitions [Cadence-15b].

If the script file specifies to read LEF files, the interconnect_mode attribute is automatically set to ple. The TCL script example uses the command to read LEF libraries. See Appendix D.

2.3.3.5 Loading the HDL Files

Use the read_hdl command for reading HDL files into RTL Compiler. When you issue a read_hdl command, RC reads the files and performs syntax checks.

To load one or more Verilog files, you can use:

read_hdl *file1*.v

read_hdl *file2*.v

read_hdl *file3*.v

Or you can load the files simultaneously:

read_hdl {*file1*.v *file2*.v *file3*.v}

The file order is important, it must respect the design file dependency hierarchy. For the design example we use:

read_hdl -v2001 {*module16_counter.v comparator_counter.v bwco.v*}

2.3.3.6 Performing Elaboration

Elaboration is required only for the top-level design. The elaborate command automatically elaborates the top-level design, and its submodules and references. During elaboration, RTL Compiler performs the following tasks: a) builds data structures; b) infers registers in the design; c) performs high-level HDL optimization, such as dead code removal; d) checks semantics.

If any gate-level netlist is included when reading the RTL files, RC automatically links the cells to their references in the technology library during elaboration. You do not have to issue an additional command for linking.

After elaboration, RTL Compiler has an internally created data structure for the whole design, enabling application of constraints and performing other operations.

The following example shows the TCL set command and the RC elaborate command for elaborating the frequency divider circuit example:

set *DESIGN bwco*

elaborate $*DESIGN*

2.3.3.7 Applying Constraints

After loading and elaborating the design, constraints must be specified [Cadence-14b]. Constraints include operating conditions, clock waveforms, and I/O timing.

Constraints can be applied in several ways: a) typed manually in the RC shell; b) included a constraints file; c) read in SDC constraints.

In the script example, the constraints are applied by reading a constraints file using the read_sdc command.

Constrains can be used to: a) define different clock signal attributes such as the duty cycle, clock skew, and clock latency; b) specify input and output delay requirements for all ports relative to clock transition; c) apply environmental attributes, such as load and drive strength for the top-level-ports; d) set timing exceptions, such as multicycle paths and false paths[37].

The following paragraphs show examples for defining clock signals.

Clocks are defined using the define_clock command. The following sentence defines a clock signal named *200MHz_CLK* with a period of 5,000 ps.

define_clock -name *200MHz_CLK* -period *5000* [clock_ports]

where clock_ports returns the input ports of the design that are clock inputs.

The clock duty-cycle can be changed by defining rising and falling edges:

define_clock –name *200MHz_CLK* –period *5000* –rise *20* –fall *80*

The slew clock attribute specifies the minimum rise, minimum fall, maximum rise, and maximum fall slew values. The following sentence sets these attributes to 250, 300, 300 and 350, respectively, for the clock signal 200MHz_CLK:

set_attribute slew {*250 300 300 350*} *200MHz_CLK*

RTL Compiler only computes timing constraints among clocks in the same clock domain. Paths between clocks in different domains are unconstrained by default.

If a clock domain is not specified, RC will assume that all the clocks are in the same domain. By default, RTL Compiler assigns clocks to domain_1, however, a personalized clock domain can be created using –domain argument together with the define_clock command. The following example shows how to create two different clocks in two separated clock domains:

[37] Timing Analysis Overview, Timing Exceptions. Feb. 14, 2019, https://www.xilinx.com/support/documentation/sw_manuals/xilinx11/ise_c_timing_analysis_overview.htm.

define_clock –domain *my_domain1* –name *clk1* –period *10000*

define_clock –domain *my_domain2* –name *clk2* –period *720*

RC enables defining other clock signal attributes such as latency and skew. In [Cadence-14b] there is a complete description of how clock properties are specified.

2.3.3.8 Applying Optimization Constraints

In addition to applying design constraints, further optimization strategies may be needed to enable the synthesis tool to achieve the desired performance goals.

RTL Compiler enable the execution of the following optimizations for: a) removing designer-created hierarchies (ungrouping); b) creating additional hierarchies (grouping); c) synthesizing a sub-design; d) creating custom cost groups for paths in the design to change the synthesis cost function. For example, the timing paths in the design can be classified into the following four cost groups: a) input-to-output paths (I2O); b) input-to-register paths (I2C); c) register-to-register (C2C); register-to-output paths (C2O).

The read_sdc command reads a constraints file in Synopsys design constraints format into RTL Compiler. It creates a cost group for each clock defined in the file. RC does not create false paths between these clocks. The design must be elaborated before reading the designs constraints.

For each path group, the worst timing path drives the synthesis cost function. See Appendix D to observe the cost groups defined in this design example.

2.3.3.9 Performing Synthesis

After setting constraints, and optimization goals, synthesis can be performed. Within RC, synthesis is performed in the following two phases: a) synthesizing the design to generic logic where technology-independent optimizations and datapath optimizations are performed; b) mapping the technology library where RC maps the generic gate-level netlist into the technology library cells and performing incremental optimization.

These two sequential steps can be performed by the synthesize command options "-to_generic" and "-to_mapped". See Appendix D for examples of synthesize and report commands.

After synthesis, the RC tool will provide a technology-mapped gate-level netlist which is

the main input for the place-and-route stage.

2.3.3.10 Reporting Synthesis Results

Encounter RTL Compiler can generate reports which allow analyzing the synthesis results. After synthesizing the design, detailed timing and area reports can be generated using several report commands: a) to generate a detailed area report, use report area; b) to generate a detailed gate selection and area report, use report gates; c) to generate a detailed timing report, including the worst critical path of the current design, use report timing.

See Appendix D for report command examples used in this design.

2.3.3.11 Writing Out Files for Place-and-Route Tool

The last step in the synthesis flow involves writing out the gate-level netlist, output SDC file, or Encounter configuration file for processing in a place-and-route tool.

By default, the write commands write output to the standard output (stdout: text terminal). For saving the information in a specific file, use the redirection symbol (>) and give the filename. Some writing out file commands are shown as follows:

For writing out the design gate-level netlist to a file called *design_netlist*.v, use:

write_hdl > *design_netlist*.v

For writing out a file that contains the timing for all modes and the design rule constraints of the design use:

write_script > *constraints*.g

The file *constraints*.g generated by this command contains the following: a) attributes related with the wire load models; b) clock objects and their reference to the pins of the design blocks; c) external_delay on all inputs and outputs; d) timing exceptions; e) max_fanout, max_capacitance and similar design rule constraints applied; and f) all user defined attributes that were created with the define_attribute command [Cadence-15a].

For writing out the design constraints in SDC format, use the write_sdc command:

write_sdc > *constraints*.sdc

To generate all files required by the Cadence place-and-route tool (Encounter Digital Implementation System EDI) use the write_encounter command:

write_encounter design *design_name* –basename *dir_name/base_name*

This sentence writes all the EDI input files to a single directory and path *dir_name/base_name* of the design *design_name*. Details and examples for this command are documented in [Cadence-15a].

2.3.3.12 Exiting RTL Compiler

There are three ways out of RTL Compiler when finishing a session: a) use the quit command; b) use the exit command; c) use the Control-c key combination twice consecutively to exit the tool immediately.

2.3.4 Logic Synthesis Results

This section presents the logic synthesis results performed by using RTL Compiler for the frequency divider circuit.

For launching RC and executing the synthesis script explained in the last section, the following sentence is typed from the command line.

rc -gui -f *synthesis_script_bwco_sdc.tcl* –log *run_log_bwco*.log

After the execution has completed, the reports and the results for each synthesis phase (generic, map, and incremental) are in the folders reports_*date_time* and outputs_*date_time*, respectively. The most important are described below.

2.3.4.1 Output Files

The design gate-level netlist file (bwco_m.v) is a structural Verilog model. It has instantiated 52 standard cells from the Virginia Tech 180 nm library. This file will be used by the place-and-route tool during the digital VLSI back-end design stage. Fig. 2.5 shows the schematic diagram generated by the RC graphic interface from the gate-level netlist file. It can be observed the circuit inputs and outputs, the standard-cell symbols and their interconnections.

The RC tool also generates others script files which are useful for configuring the place-and-route tool in the back-end design stage, as is explained in Section 2.3.3.11 of this document.

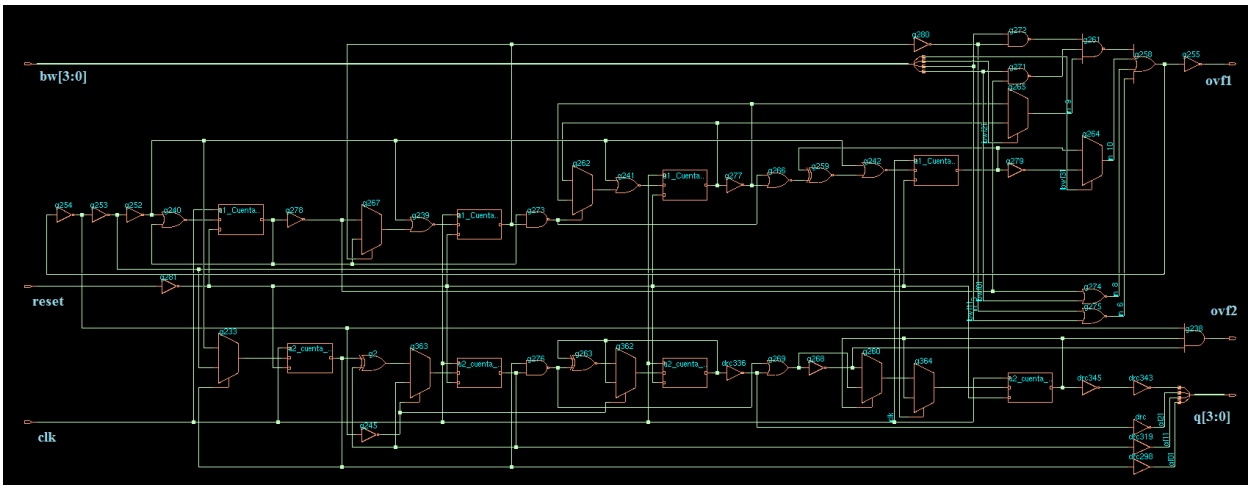

Fig. 2.5 Schematic diagram of frequency divider circuit (bwco) generated by RTL Compiler. This is the graphical view of the gate-level netlist for the synthesized circuit example.

See Section K of Appendix D for the specific script files generated for this design example.

2.3.4.2 Synthesis Results Reports

The RTL Compiler synthesis tool can generate several reports: timing, silicon area, and power consumptions. Fig. 2.6 to Fig. 2.8 show report summaries for the incremental synthesis phase of the design example.

Fig. 2.6 shows the standard cells and area report, bwco_gates.rpt. The design example uses different gates, flip-flops, and combinational blocks. The required area by each cell is specified, as well as the library to which the cell belong. The synthesized frequency divider has 44 combinational and eight sequential cells. The total estimated area is 3,812 μm^2.

The quality of silicon report, final.rpt, is shown in Fig. 2.7. This shows the evolution of the logic synthesis through the generic, map, and incremental phases: the changes in the required area and standard cells are reported. Also, the critical-path slacks for the defined cost groups are listed.

Finally, Fig. 2.8 shows the quality of results report, bwco_qor.rpt, which contains the path slacks and the total negative slack for the created cost groups, as well as the estimated power consumption and the maximum and minimum fanout.

```
============================================================
  Generated by:           Encounter(R) RTL Compiler v12.10-s012_1
  Generated on:           Nov 12 2014  01:20:32 pm
  Module:                 bwco
  Technology libraries:   vtvt_tsmc180
                          physical_cells
  Operating conditions:   _nominal_
  Interconnect mode:      global
  Area mode:              physical library
============================================================


  Gate   Instances    Area       Library
-----------------------------------------------
and3_1           1    55.112    vtvt_tsmc180
buf_1            1    45.927    vtvt_tsmc180
buf_2            1    45.927    vtvt_tsmc180
drp_1            8  1763.597    vtvt_tsmc180
inv_1           11   303.118    vtvt_tsmc180
inv_2            4   110.225    vtvt_tsmc180
mux2_1           9   661.349    vtvt_tsmc180
nand2_1          4   146.966    vtvt_tsmc180
nand3_1          1    45.927    vtvt_tsmc180
nor2_1           7   257.191    vtvt_tsmc180
or2_1            1    45.927    vtvt_tsmc180
or4_1            1    64.298    vtvt_tsmc180
xnor2_1          2   183.708    vtvt_tsmc180
xor2_1           1    82.669    vtvt_tsmc180
-----------------------------------------------
total           52  3811.941


   Type    Instances   Area   Area %
--------------------------------------
sequential         8 1763.597   46.3
inverter          15  413.343   10.8
buffer             2   91.854    2.4
logic             27 1543.147   40.5
--------------------------------------
total             52 3811.941  100.0
```

Fig. 2.6 Gates report (bwco_gates.rpt): standard-cells used and required area for the frequency divider circuit.

```
Working Directory = /home/usuario/Cuauh/Cadence/my_designs/bwco/RC_synthesis
QoS Summary for bwco
================================================================================
Metric                  generic    map        incremental
================================================================================
Slack (ps):               2235.0     1978.4     1934.3
  R2R (ps):               3792.3     3396.8     3461.8
  I2R (ps):               3110.0     2738.3     2803.3
  R2O (ps):               2929.6     2635.9     2592.8
  I2O (ps):               2235.0     1978.4     1934.3
  CG  (ps):             no_value   no_value   no_value
TNS (ps):                       0          0          0
  R2R (ps):                     0          0          0
  I2R (ps):                     0          0          0
  R2O (ps):                     0          0          0
  I2O (ps):                     0          0          0
  CG  (ps):             no_value   no_value   no_value
Failing Paths:                  0          0          0
Area:                        2998       3950       3812
Instances:                     43         58         52
Utilization (%):             0.00       0.00       0.00
Tot. Net Length (um):   no_value   no_value   no_value
Avg. Net Length (um):   no_value   no_value   no_value
Total Overflow H:               0          0          0
Total Overflow V:               0          0          0
Route Overflow H (%):   no_value   no_value   no_value
Route Overflow V (%):   no_value   no_value   no_value
================================================================================
CPU  Runtime (m:s):          00:01      00:01      00:00
Real Runtime (m:s):          00:06      00:00      00:01
CPU  Elapsed (m:s):          00:04      00:05      00:05
Real Elapsed (m:s):          00:07      00:07      00:08
Memory (MB):                 61.16      64.23      67.30
================================================================================

================================================================================
Flow Settings:
================================================================================
Total Runtime (m:s): 00:07
Total Memory (MB):   67.30
Executable Version:  12.10
================================================================================
```

Fig. 2.7 Quality of silicon report (final.rpt): evolution of timing, cell numbers, and required area for each synthesis stage.

```
================================================================
  Generated by:           Encounter(R) RTL Compiler v12.10-s012_1
  Generated on:           Nov 12 2014  01:20:32 pm
  Module:                 bwco
  Technology libraries:   vtvt_tsmc180
                          physical_cells
  Operating conditions:   _nominal_
  Interconnect mode:      global
  Area mode:              physical library
================================================================
Timing
--------
   Clock   Period
------------------
200MHz_CLK 5000.0

   Cost      Critical       Violating
   Group    Path Slack TNS    Paths
---------------------------------------
default       No paths   0
I2C             2803.3   0          0
C2O             2592.8   0          0
C2C             3461.8   0          0
I2O             1934.3   0          0
---------------------------------------
Total                    0          0

Instance Count
--------------
Leaf Instance Count            52
Sequential Instance Count       8
Combinational Instance Count   44
Hierarchical Instance Count     0

Area & Power
------------
Total Area                          3811.941
Cell Area                           3811.941
Leakage Power                       0.024 uW
Dynamic Power                       619.504 uW
Total Power                         619.528 uW

Max Fanout                          8 (n_49)
Min Fanout                          1 (n_0)
Average Fanout                      2.0
Terms to net ratio                  2.8
Terms to instance ratio             3.1
Runtime                             5 seconds
RC peak memory usage:               67.30
EDI peak memory usage:              no_value
Hostname                            cuauhtemoccad
```

Fig. 2.8 Quality of result report (bwco_qor.rpt): summary of critical path slack, total negative slack of cost-groups, used standard cells, total area and estimated power consumption.

2.4. Conclusions

In this chapter, the basic components needed for implementing digital VLSI front-end design at ITESO have been identified. The MOSIS IC fabrication processes for education and research institutions were identified. The IBM 130 nm (8RF-DM) technology was selected as the most suitable for manufacturing interesting IC designs, for research projects in the context of the Doctoral Program in Engineering Sciences at ITESO. The PDK and standard-cell libraries concepts were presented, which are fundamental components for implementing digital VLSI design at ITESO. Furthermore, the steps for synthesizing a sequential circuit using RTL Compiler have been documented, which are fundamental in the digital VLSI design flow. A Cadence recommended logic synthesis flow was implemented and verified by synthesizing a frequency divider circuit. The gate-level netlist based on a specific standard-cell technology (180 nm Virginia Tech Library) was generated. This is the main input for the place-and-route tool in the back-end design stage of a digital VLSI circuit. The RC synthesis process requires a script file with the attributes and commands for controlling synthesis results and output reports. The creation and content of the synthesis script were explained. Several synthesis reports such as quality of silicon and quality of results were presented. They show the outcomes of synthesizing a frequency divider circuit, for example, the kind of standard-cells used, required silicon area, timing, and power consumption.

The subjects here documented, allowed to implement for the first time at ITESO integrated circuit laboratory, the logic synthesis of a digital circuit using RC Cadence tool, and the VTVT standard-cell library.

3. On-Chip Implementation of Low-Latency Bit-Accurate Fixed-Point RSR Unit

Many popular applications, such as gaming, digital signal processing, and communications systems, require computation of the reciprocal square root operation (RSR). Although several architectures have been reported for computing the RSR operation, these are mainly focused on accelerating high-precision floating-point units. However, in low-power mobile-device implementations, fixed-point (FxP) units are preferred due to their low-computational cost and power consumption. This chapter presents an on-chip implementation of a low-latency, bit-accurate, RSR IP-Core using FxP arithmetic and a 130 nm CMOS ASIC technology.

3.1. Relevance of Reciprocal Square Root and Previous Works

The reciprocal square root (RSR) is a fundamental and recurrent operation in digital signal processing (DSP) algorithms where matrix decomposition and solution of systems of linear equations are required. For example, in multiple-input multiple-output (MIMO) wireless communication to perform tasks such as digital modulation [Chen-13], channel estimation [Salmela-06], [Salmela-11], singular-value decomposition [Markovic-07], and matrix inversion [Mahapatra-12]. Likewise, in the area of digital signal processing these nonlinear operations are required for matrix decomposition[Singh-07], [Luethi-08], [Liu-17a], [Liu-17b]. Furthermore, the RSR operation is required for 3D-image rendering in gaming applications [Woo-09], [Kim-11].

Silicon IP-cores can be utilized to improve the performance of electronic applications implemented in low-power embedded systems and mobile devices with limited computational resources, for example, the NXP microcontroller based on ARM Cortex-M4[38]. In this kind of systems, the processing unit could present bottlenecks produced by complex operations, such as the following elementary functions: exponential, logarithms, trig, hyperbolic trig, roots, RSR,

[38] NXP Processors and Microcontrollers, Low-power 32-bit Microcontrollers. Nov. 11, 2017, https://www.nxp.com/products/processors-and-microcontrollers/arm-based-processors-and-mcus/kinetis-cortex-m-mcus:KINETIS.

among others. In these applications, it is of paramount importance to reduce the microprocessor load, by implementing the complex operations in silicon IP-cores instead of executing them by software instructions. These customized IP-cores improve overall system performance in terms of area, speed, and power consumption [Liu-17a], [Liu-17b], [Woo-09], [Kim-11], [Markovic-06], [Markovic-07]. Due to its ever-expanding presence, having an off-the-shelf RSR silicon IP-core reduces time-to-market cycles and increases resource utilization.

Several double-precision floating-point (FP) architectures for computing the RSR operation have been proposed. In [Takagi-01] and [Lang-03], a modified digit-recurrence algorithm is used, leading to high-latency (28 cycles). Initial works [Wong-94] used an architecture based on rectangular multipliers. Later, [Ercegovac-00] showed improved performance when using smaller multipliers and Taylor series evaluations. The proposal in [Piñeiro-02] presents the best estimated cost-delay tradeoff among those mentioned here. It is based on look-up tables (LUTs), polynomial approximation and one Goldschmidt iteration. These architectures focus mainly on accelerating high-precision FP units. Hence, they are not suitable for low-power, low-cost mobile devices due to the hardware cost and power consumption.

Furthermore, FP single-precision designs for computing the square-root operation have been reported. In [Ren-93] and [Srinivas-95], shared divider/square-root designs are reported, the integrated-circuit layouts are shown, and the area and delay are specified, however, measurements of the manufactured chips are not reported. Moreover, the technologies (1.2 μm) and design methodologies used in [Ren-93] and [Srinivas-95] are far below the state-of-the-art. A standard-cell implementation of the RSR based on LUTs and a modified Newton-Raphson (NR) iteration is presented in [Schulte-99]. An improved version of [Schulte-99] was later proposed in [Wires-06]. Alternatively, [Kwon-08] reports a standard-cell implementation of the square root based on LUTs and Taylor series. Synthesis results from a digit-recurrence square-root circuit for two standard-cell technologies (40 and 60 nm) are presented in [Suresh-13], which reports an estimated power consumption for each technology. A digit-recurrence implementation for computing the $1/x$, $\sqrt{x}$, and $1/\sqrt{x}$ operations is presented in [Butts-11]; it is based on radix-8 for determining the next digit and shows a latency of eight cycles.

In real mobile applications, the high-demand computing tasks are implemented in specialized FxP units. This leverages lower hardware cost and reduces the power consumption of the FxP implementations [Khirallah-03], [Wang-10], [Salmela-11]. Examples of this trend are the

applications presented in [Salmela-06], [Singh-07], [Liu-17a], and [Liu-17b], all of which use 16-bit FxP units to compute either the square-root or the RSR operation. Similarly, [Luethi-08] and [Sohn-06] report the use of 23-bit and 32-bit FxP units to perform the same operations, respectively.

Despite the advantages of the FxP arithmetic for real applications on mobile devices, few papers have reported an FxP implementation, either of the square-root [Wang-10], [Sajid-12], [Martin-Del-Campo-12], [Seth-11] or the RSR [Salmela-11], [Pizano-Escalante-15], [Rounioja-03].

In the following section, the algorithm to implement the RSR operation is introduced, it is based on the NR method and a piecewise-polynomial approximation in a reduced range. The algorithm is patented by the authors in [Parra-Michel-18], [Pizano-Escalante-15]. Hereafter the algorithm is named 2C-RSR, which is presented for being implemented as a silicon IP using FxP arithmetic on a 130 nm ASIC technology.

3.2. 2C-RSR Algorithm

The 2C-RSR algorithm computes the operation

$$y = 1/\sqrt{x} \tag{3-1}$$

where $x, y \in \mathbb{R} \mid x, y = \sum_{i=-f}^{k-1} b_i 2^i$ with $b_i \in \{0,1\}$, and $k, f \in \mathbb{Z}$ are the number of bits for representing the integer and fractional parts, respectively, of *x* and *y* in FxP format.

In this thesis document, an FxP format is represented by notation *Q*(*w*, *f*, *sign*), where $w = k + f$ is the word-length and $sign \in \{s, u\}$ indicates signed or unsigned format, respectively. Due to the finite size of *w* in real implementations, the result computed by (3-1) is an approximation of the exact value, i.e., $1/\sqrt{x}$ computed using infinite precision. Nevertheless, the 2C-RSR algorithm is able to provide a result with a maximum error of $2^{-f}/2$, which makes the result bit-accurate with respect to the result computed by a double-precision FP unit (IEEE 754-2008 standard) [IEEE-08] when this is represented in the selected *Q*(16,11,*u*) FxP format. We selected this format because it allows representing the magnitude of standard-Gaussian random variables, which is useful to study real-valued random variables whose distributions are unknown

TABLE 3.1. BIT-ACCURATE ASSERTION

$\Theta = 1/\sqrt{9}$

Value v_x	$Q_{16}^{11}\{v_x\}$	$Q_{20}^{15}\{v_x\}$
v_{FP} = 0.333333333333333	0.01010101011	0.010101010101011
v = 0.33349609375	0.01010101011	0.010101010110000
Is v bit-accurate?	yes	no

[Williams-16].

3.2.1 Bit-Accurate Property

Since bit-accurate is not a standardized concept, we define it below as used in this thesis document.

The conversion operation of v, from decimal to binary FxP format, is

$$Q_w^f\{v\} = v^{w,f} \tag{3-2}$$

where v is the decimal representation of the result obtained from any arithmetic operation Θ performed by an FxP arithmetic unit. The expression $v^{w,f}$ stands for the binary representation of v in FxP format considering w and f parameters. Likewise, the conversion operation of v_{FP}, from decimal to binary FxP format, is

$$Q_w^f\{v_{FP}\} = v_{FP}^{w,f} \tag{3-3}$$

where v_{FP} denotes the decimal representation of the result performed by a double-precision FP arithmetic unit, and $v_{FP}^{w,f}$ is the binary representation of v_{FP} in FxP format. Therefore, the bit-accurate property holds for v when (3-4) is met,

$$v^{w,f} = v_{FP}^{w,f}. \tag{3-4}$$

To illustrate the bit-accurate property, Table 3.1 shows the comparison of two numerical results. The first row shows the result of the $1/\sqrt{9}$ operation performed by an FP arithmetic unit and its equivalent value when this is represented in FxP format (which can be obtained by using the *fi*(v,0,w,f) Matlab function). The second row shows the result of the same operation performed by a bit-accurate FxP arithmetic unit using w = 16 and f = 11. When this result is represented in

Q(16,11,*u*) format, all the bits are equal to the corresponding $v_{FP}^{16,11}$ value, and it can be said that the obtained result is bit-accurate. It must be noted that this does not necessarily holds for a different format, *Q*(20,15,*u*) in this example. The advantage of a bit-accurate result computed by an FxP unit is that the result can be shared with a more-precise FP unit without introducing a conversion error.

3.2.2 Top-level Description of the 2C-RSR Algorithm

The 2C-RSR algorithm reported in [Pizano-Escalante-15] is based on the NR method. The seed for the NR iteration is computed by a piecewise-polynomial approximation. Due to the nonlinearity of the RSR function, the polynomials are evaluated in a reduced range of *x*, namely the reduced range (*rr*). This condition improves the polynomial fit and results in a better approximation. For computing the RSR of *x* when *x* is outside *rr* range, a scaling and a de-scaling step are required. At the end, a rounding step is applied to obtain a *bit-accurate* result with a maximum error of ½ unit in the last place (ulp), with $\text{ulp} = 2^{-f}$ for the *Q*(*w*, *f*, *sign*) format. Each step of the algorithm is summarized below.

3.2.3 Newton-Raphson Method

The Newton-Raphson iteration for computing the RSR operation [Ercegovac-05] is obtained by applying the general NR equation

$$y_{i+1} = y_i - f(y_i)/f'(y_i) \tag{3-5}$$

to the function $f(y) = 1/y^2 - z$, where $f'(y)$ denotes the first derivative and $z \in wr$ represents the scaled value of *x* defined by

$$z = x/2^{2n}, \tag{3-6}$$

where $n \in \mathbb{Z}$ is the scaling exponent, and 2^{2n} is the scaling factor. From the foregoing, the NR iteration for the RSR function is defined by

$$y_{i+1} = (y_i/2)(3 - zy_i^2), \tag{3-7}$$

where y_i represents the *i*th iteration, which converges quadratically to $1/\sqrt{z}$ for any seed

$y_0 \in \left(0, \sqrt{3}/\sqrt{z}\right)$ [Joldes-16]. The quadratic convergence of (3-7) is observed from the error relation between two consecutive iterations and is defined by

$$\varepsilon_{i+1} = \varepsilon_i^{\,2}\left(-zy_i/2 - \sqrt{z}\right), \tag{3-8}$$

where $\varepsilon_i = y_i - 1/\sqrt{z}$ is the NR approximation error of the *i*th iteration and $\varepsilon_{i+1} = y_{i+1} - 1/\sqrt{z}$ is the approximation error of the next iteration, which is roughly the square of the *i*th-iteration error. Due to the quadratic convergence of (3-7), the number of correct bits of y_{i+1} roughly doubles on each iteration [Ercegovac-05].

3.2.4 Seed Computation

In the proposed architecture, the NR method is provided with a seed value y_0 close to $1/\sqrt{z}$ by a piecewise-polynomial approximation. Since the piecewise approximation is performed for a limited range of *x* (*rr*), the polynomial grade is reduced [Ercegovac-04], resulting in a better polynomial fitting for the RSR function, and helps to meet the approximation error requirements. The first approximation of $1/\sqrt{z}$ is obtained by using a polynomial defined by

$$y_0 = \sum_{i=0}^{M} a_i z^i, \tag{3-9}$$

where a_i are the polynomial coefficients and *M* is the polynomial's degree. In order to apply the piecewise-polynomial approximation, the following factors are considered: the error objective, the *rr* range, the number of subintervals into which *rr* range is divided, the polynomials' degree, and the hardware-implementation cost.

3.2.5 Scaling and De-scaling Operations

A good feature of the 2C-RSR algorithm is the scaling step, as this allows improving the polynomial approximation and the algorithm architecture for the target *rr* range. In the scaling step, the range of *x* is divided into several intervals, each of which is linearly mapped into the reduced range *rr* using different scaling exponent *n*. From (3-1) and (3-6), the scaling effect yields

$$y = \frac{1}{2^n \sqrt{x/2^{2n}}} = \left(2^{-n}\right)\left(1/\sqrt{z}\right). \tag{3-10}$$

Since (3-7) converges to $1/\sqrt{z}$, an approximation of y is obtained by

$$\tilde{y} = 2^{-n} y_i, \tag{3-11}$$

where 2^{-n} represents the de-scaling factor.

3.2.6 Rounding Operation

In the rounding step, a round to nearest operation[39] [Ercegovac-04], with a tie-breaking rule is applied to the de-scaled approximation $\tilde{y}$ obtained using (3-11). Thus, a bit-accurate RSR result is obtained in $Q(w, f, sign)$ format, with an error of $2^{-f}/2$ as the maximum. This operation is expressed by

$$y = RNQ_w^f\{\tilde{y}\}. \tag{3-12}$$

3.3. 2C-RSR Hardware Architecture

This section describes the hardware architecture of the 2C-RSR algorithm. Each step of the algorithm is implemented by hardware modules, which are shown in Fig. 3.1. The proposed architecture computes the RSR using the $Q(16,11,u)$ format, a new bit-accurate result is produced in only two clock cycles. The first cycle is used to compute the seed, and the second one to perform the NR iteration. Therefore, in order to meet the maximum error condition, a seed with eight-bit of accuracy must be fed to the NR iteration.

The names of the modules represented in Fig. 3.1 and the operation that implement are: the encoder, ENC, and barrel shifter, BS1, perform the scaling operation; the polynomial and Newton-Raphson, PNR, implement the seed computation and NR iteration; the ENC and BS2 perform the de-scaling operation; the RND module implements the rounding operation; the control unit, CU, synchronizes the functions of the 2C-RSR modules; the overflow detector, OVD, and saturation,

[39] EE Times, An introduction to different rounding algorithms. Sep. 30, 2017, http://www.eetimes.com/document.asp?doc_id=1274485.

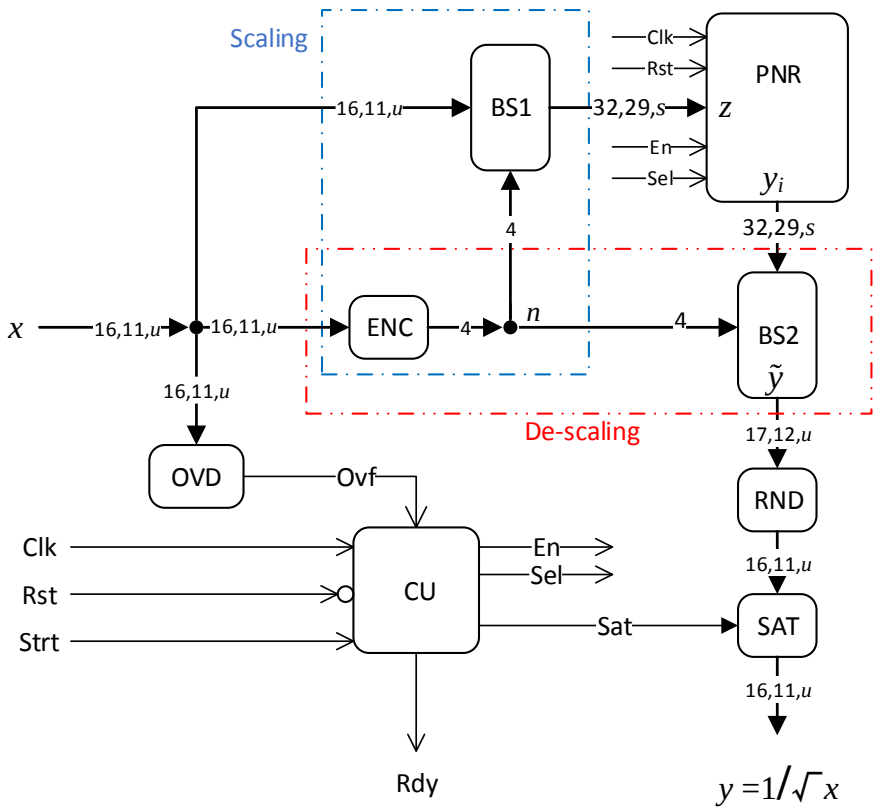

Fig. 3.1 Hardware architecture of the 2C-RSR algorithm.

SAT, modules are for signaling that the result cannot be represented in the $Q(w, f, sign)$ format.

In the following sections, each module and its design considerations are described.

3.3.1 Overflow Detector and Control Unit

Before computing the RSR operation, it is necessary to detect any overflow conditions, which depend on the selected FxP format. For the $Q(16,11,u)$ format, the 2C-RSR architecture produces an overflow when $x \leq 2^{-10}$. If an overflow occurs, the overflow detector (OVD in Fig. 3.1) triggers the Ovf signal to indicate to CU that y must be assigned to the maximum-representable value by the FxP format. This is performed in the saturation block SAT. The CU input signals are: the reset Rst that establishes the initial conditions; the clock Clk for controlling the timing in the circuit; and the Strt that commands the start of a new operation. The CU outputs are the selection Sel and enable En signals used to control the PNR module shown in Fig. 3.2.

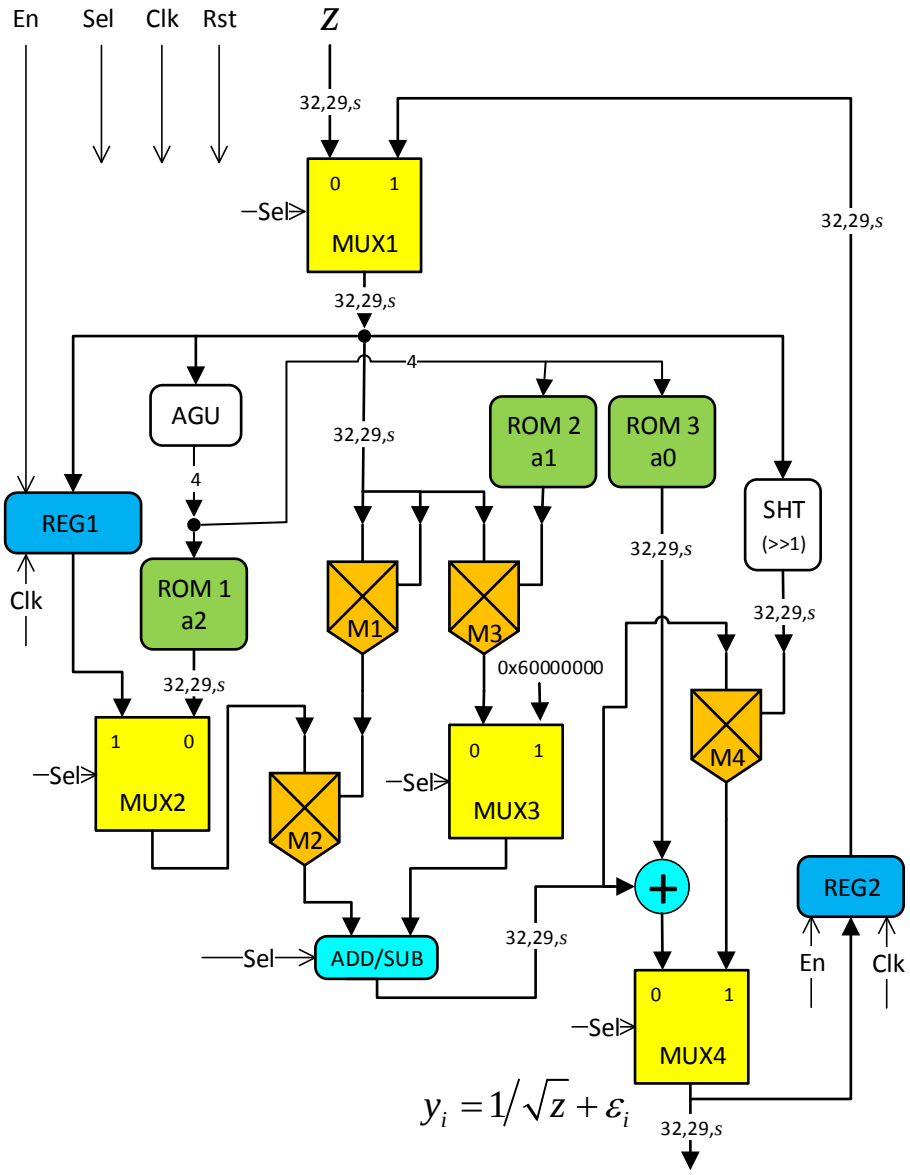

Fig. 3.2 Hardware architecture of the PNR module that performs the piecewise-polynomial-seed computation and the NR iteration. The seed computation is carried out in the first clock cycle (Sel = 0, En = 1) whereas the NR evaluation is done in the second clock cycle (Sel = 1, En = 0).

3.3.2 Scaling Module

The RSR architecture proposed in this thesis document is nearly agnostic to the *x* range. The scaling module not only improves the polynomial fitting but also enables this feature. For

TABLE 3.2. SCALING EXPONENTS AND INTERVALS OF x

$\alpha = 0.5$, $\beta = 2$, $w = 16$, $k = 5$, $f = 11$

Scaling Exponent n	X_{LBn}	X_{UBn}
2	8	31.99951171875
1	2	7.99951171875
0	0.5	1.99951171875
-1	0.125	0.49951171875
-2	0.03125	0.12451171875
-3	0.0078125	0.03076171875
-4	0.001953125	0.00732421875

example, if a different FxP format is required, this change can be addressed by modifying the scaling and descaling modules accordingly, and representing in the new required FxP format the MUX3 input constant, which is the "3" number in (3-7).

From (3-6) and (3-11), the scaling and de-scaling operations can be implemented by bit-shifting of x and y_i, respectively. Each n value transforms an x interval $[x_{LBn}, x_{UBn}]$ into the rr range. The interval bounds are defined by

$$x_{LBn} = \alpha 2^{2n},\ x_{UBn} = \beta 2^{2n} - 2^{-f},\ -f \le 2n \le k\,. \tag{3-13}$$

In the proposed architecture, the values of α and β impact: a) the piecewise polynomial-approximation error; b) the required number of subintervals (polynomials) in which the rr range is to be divided in order to meet the target error; and c) scaling and de-scaling operations. In this case, the values of $\alpha = 0.5$ and $\beta = 2.0$ are obtained by using piecewise-polynomial approximation for the RSR function. Furthermore, α and β values are selected to be powers of two, to achieve an efficient implementation of the scaling and de-scaling modules.

The selected input and output FxP format is $Q(16,11,u)$, i.e., $w = 16$ bits, $f = 11$ bits, $sign = u$, and therefore $k = 5$ bits. This format allows representing the magnitude of standard-Gaussian random variables, which is useful to study real-valued random variables whose distributions are unknown. Once the variables α, β, f, and k are defined, the intervals of x are calculated using (3-13) ; these are given in Table 3.2.

The scaling operation is performed as follows. If there is no overflow condition, the encoder ENC and barrel shifter BS1 modules scale the input x to the reduced range rr, according to

(3-6). The ENC provides the correct scaling exponent *n*, depending on the interval in which *x* is located. The scaling exponents and *x* intervals are reported in Table 3.2. The ENC is a logic comparator implemented by defining the bounds of scaling intervals as powers of two. Thus, the comparator uses the 14 most significant bits of *x* to provide *n*. The BS1 performs the operation defined by (3-6), shifting *x* by 2*n*-bits in the direction given by the sign of *n*. Moreover, to avoid losing seed-computation accuracy, 16 bits are concatenated to the fractional part of the shifted *x*, increasing its precision for computing the NR seed using the *Q*(32,29,*s*) format.

3.3.3 Polynomial Approximation and NR Module (PNR)

To ensure a seed with eight-bit of accuracy from the polynomial approximation, an FxP analysis is performed by using the DSP techniques proposed in [Sung-95], [Menard-05]. This analysis shows that the word format for computing the seed and y_i needs to be *Q*(32,29,*s*).

To reduce hardware complexity for implementing the polynomial approximation and NR method, we choose two as the polynomials' degree. The parameters for the piecewise-polynomial approximation are computed using the minimax and least-square error criteria [Muller-05]. In both cases, the required seed with eight-bit of accuracy is achieved using 14 polynomials. The unequal subintervals associated to the polynomials into which *rr* range is divided are shown in Table 3.3. To minimize the polynomial-selection hardware, the subinterval boundaries rr_{LB} and rr_{UB} are defined to be accurately represented by the selected FxP format. Table 3.4 shows the polynomial coefficients computed using the least-square approximation.

It is important to note that the number of subintervals and polynomial coefficients does not change when another FxP format is required; it is only necessary to apply a new scaling exponent *n*, as discussed in Section 3.3.2.

The seed computation (3-9) and NR iteration (3-7) steps both perform a quadratic-polynomial evaluation. We take advantage of this to reduce silicon area by implementing both operations in the same module PNR, as shown in Fig. 3.2. The architecture of the PNR module is composed of three read-only memories ROM1-ROM3 for storing the polynomial coefficients, one address-generator unit AGU, four multipliers M1-M4, two registers REG, one combinational shifter SHT, four multiplexers MUX, one adder/subtractor ADD/SUB, and one adder. All of these modules use the *Q*(32,29,*s*) format.

TABLE 3.3. BOUNDS OF THE 14 SUBINTERVALS IN WHICH *rr* RANGE IS DIVIDED FOR THE PIECEWISE-POLYNOMIAL APPROXIMATION

Subinterval	rr_{LB}	rr_{UB}
1	0.5	0.513671875
2	0.513671875	0.529296875
3	0.529296875	0.5537109375
4	0.5537109375	0.5849609375
5	0.5849609375	0.62890625
6	0.62890625	0.6875
7	0.6875	0.75
8	0.75	0.8125
9	0.8125	0.9375
10	0.9375	1.125
11	1.125	1.3125
12	1.3125	1.5
13	1.5	1.75
14	1.75	2

The operation of the PNR module shown in Fig. 3.2 is executed in two clock cycles. In the first cycle, the seed computation is performed as follows. The CU module provides the signals Sel and En to configure the data path (MUX, REG, and ADD/SUB) to evaluate (3-9). With Sel = 0 and En = 1, the scaled data *z* is connected to the input of the modules AGU, M1, M3, SHT, and REG1. The AGU generates the ROM addresses of the polynomials' coefficients depending on the subinterval to which z belongs according to Table 3.3. The AGU is a priority encoder, where the bounds of *rr* subintervals are defined as powers of two. M1 computes the quadratic term z^2 of (3-9), M2 computes $a_2 z^2$ and M3 computes $a_1 z$. The last two products are added in the ADD/SUB block. The outputs of ADD/SUB block and ROM3 (a_0) are added to complete the seed calculation (y_0). The former is stored in REG2.

TABLE 3.4. FLOATING-POINT, °, AND FIXED-POINT, ¤, COEFFICIENTS FOR THE PIECEWISE-POLYNOMIAL APPROXIMATION

N: number of subinterval

N	T	a_2	a_1	a_0
1	°	2.070310683729	-3.490936206251	2.642141527551
	¤	0x423FFC30	0x904A4027	0x548C6C63
2	°	1.927325916268	-3.344069470610	2.604519952278
	¤	0x3DACA766	0x94FD6205	0x53583A3A
3	°	1.754197082325	-3.160906134777	2.556195569455
	¤	0x382261EB	0x9AD9DB60	0x51CC5AA6
4	°	1.547088157005	-2.931440188928	2.492793983503
	¤	0x3181BF05	0xA231A458	0x4FC4F7E3
5	°	1.320898869568	-2.666469549047	2.415382966010
	¤	0x2A44CDB4	0xAAAC480D	0x4D4AD137
6	°	1.077967960168	-2.360542638366	2.319293191251
	¤	0x227EB6A9	0xB4766F48	0x4A37A65A
7	°	0.965936441256	-2.213950778620	2.271577691221
	¤	0x1EE8F38A	0xB92750B2	0x48B0C3B2
8	°	0.700440349993	-1.821489125616	2.126890448060
	¤	0x166A01E1	0xC5B65C6F	0x440F7C8E
9	°	0.529134728524	-1.540725263357	2.012053720623
	¤	0x10EEABF4	0xCEB260EE	0x4062BE7B
10	°	0.351737406365	-1.206434709560	1.854793408661
	¤	0x0B416ECE	0xD964E309	0x3B5A77B4
11	°	0.230822482097	-0.936210142810	1.703970519464
	¤	0x0762E5D4	0xE20A9106	0x3686ED2E
12	°	0.161037227787	-0.7539341201073	1.585033998371
	¤	0x0527378B	0xE7DFC58D	0x32B89938
13	°	0.112221918030	-0.607157667474	1.474757219720
	¤	0x0397526B	0xEC922A15	0x2F31360D
14	°	0.078327429971	-0.489116629813	1.372019228973
	¤	0x0281A886	0xF0592814	0x2BE794DE

The NR method starts with the rising edge of the second clock cycle, storing z in REG1, y_0 in REG2, setting Sel = 1, and En = 0. At this point, the architecture datapath is configured to

evaluate (3-7): the MUX1 output is y_0, M1 computes y_0^2 and SHT generates $y_0/2$. MUX2 selects z and M2 multiplies it by y_0^2. MUX3 selects the constant 0x60000000 (+3 in $Q(32,29,s)$ format). ADD/SUB performs $3 - zy_0^2$ with the outputs of MUX3 and M2. Finally, M4 multiplies the outputs of the SHT and ADD/SUB modules. The output of MUX4, y_i, is the NR approximation of $1/\sqrt{z}$ in $Q(32,29,s)$ format.

3.3.4 De-scaling Module

As shown in Fig. 3.1, we reuse the ENC module, and the other barrel shifter BS2 to perform the de-scaling step. With n already determined, BS2 performs (3-11) shifting y_i by n bits in the opposite direction of the sign of n. The result $\tilde{y}$ is truncated into $Q(17,12,u)$ format to preserve the rounding bit.

3.3.5 Rounding Module

To achieve a bit-accurate result, the output of BS2, $\tilde{y}$, should be rounded. For this reason, the de-scaling output is one-bit wider than the required output format $Q(16,11,u)$. The RND module performs the round to nearest operation (round-half-up)[40]. Using the least significant bit (LSB) of $\tilde{y}$, the following criterion is taken: if LSB = 0, the result is correct in 16 bits, otherwise, 2^{-11} is added to the result to achieve the bit-accurate result y in $Q(16,11,u)$ format.

3.4. 2C-RSR ASIC Implementation and Results

3.4.1 ASIC Implementation

The Verilog register-transfer-level (RTL) model of the 2C-RSR architecture is verified by implementing a test bench in the Mentor Graphics® ModelSim environment. All the valid input

[40] EE Times, An introduction to different rounding algorithms. Sep. 30, 2017, http://www.eetimes.com/document.asp?doc_id=1274485.

values are applied to the design, and the outputs are exhaustively compared with the corresponding golden values, which are computed by using the double-precision-RSR operation and representing the results in $Q(16,11,u)$ format. Checkers and monitors are implemented in the test bench in order to guarantee that the circuit model achieved the bit-accurate property.

The verified RTL model is synthesized and implemented using the ARM standard-cell library for the Globalfoundries 130 nm CMOS (8RF-DM) process. The logic and physical synthesis are performed with RTL Compiler and Encounter Digital Implementation, respectively, both tools from Cadence®. In addition, the netlist model passed the logic-equivalence check and the static-timing-analysis tests.

The post-placed and routed netlist of the 2C-RSR chip is verified using the same test bench as that one used for verifying its RTL model. The timing and the functionality are correct.

The chip layout-versus-schematic (LVS) and design-rule-checking (DRC) tests are performed using the Virtuoso and Calibre tools. The final 2C-RSR physical design is shown in Fig. 3.3. The standard-cell area is 0.2289 mm^2, and the total chip area including the pad-frame is 1.598 mm^2. The 2C-RSR is manufactured by the MOSIS MEP program using the 130 nm CMOS (8RF-DM) process. The chip microphotograph is shown in Fig. 3.4.

The pin count of the 2C-RSR chip is as follows (see Fig. 3.5): the 16-bit input, $x[15:0]$, Start input, Reset input, clk input, the 16-bit output, $y[15:0]$, READY output, 1.2V core power supply (VDD and VSS) and 2.5V input-output (I/O) power supply (DVDD and DVSS). We decided to dispense with the overflow signal in order to implement the chip in a DIP40 package.

The functional and timing tests of the 2C-RSR chip are carried out with a test platform [Aguilera-Galicia-16] based on an FPGA board (Xilinx Spartan-6), a mixed-signal oscilloscope (Keysight MSO9064A), and a logic analyzer (Keysight 16862A), as shown in Fig. 3.6. The test vectors are sequentially and exhaustively applied by a finite-state machine inside the FPGA. Full coverage of test cases is achieved and the input-output pairs of the chip for several frequencies are acquired with the logic analyzer.

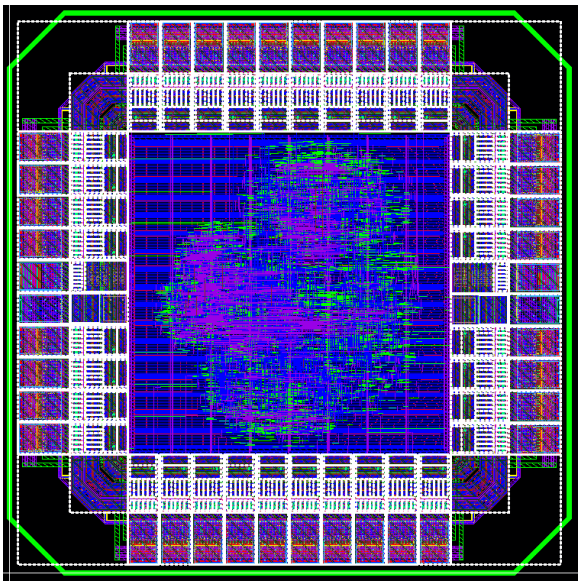

Fig. 3.3 Physical design of the 2C-RSR integrated circuit.

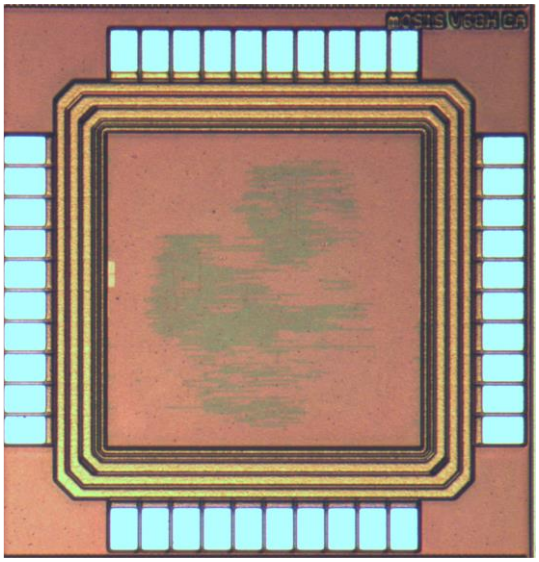

Fig. 3.4 The 2C-RSR integrated circuit microphotograph.

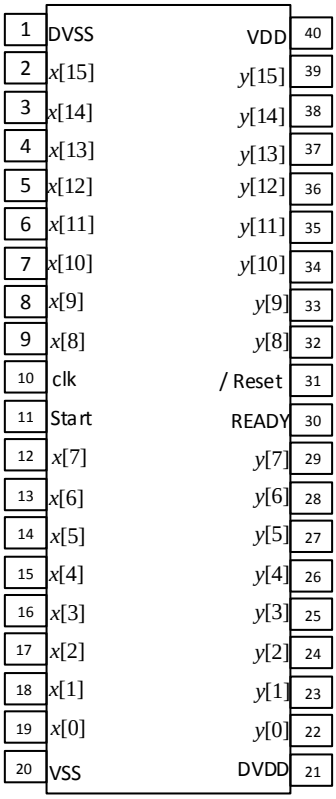

Fig. 3.5 Pin layout of the 2C-RSR integrated circuit.

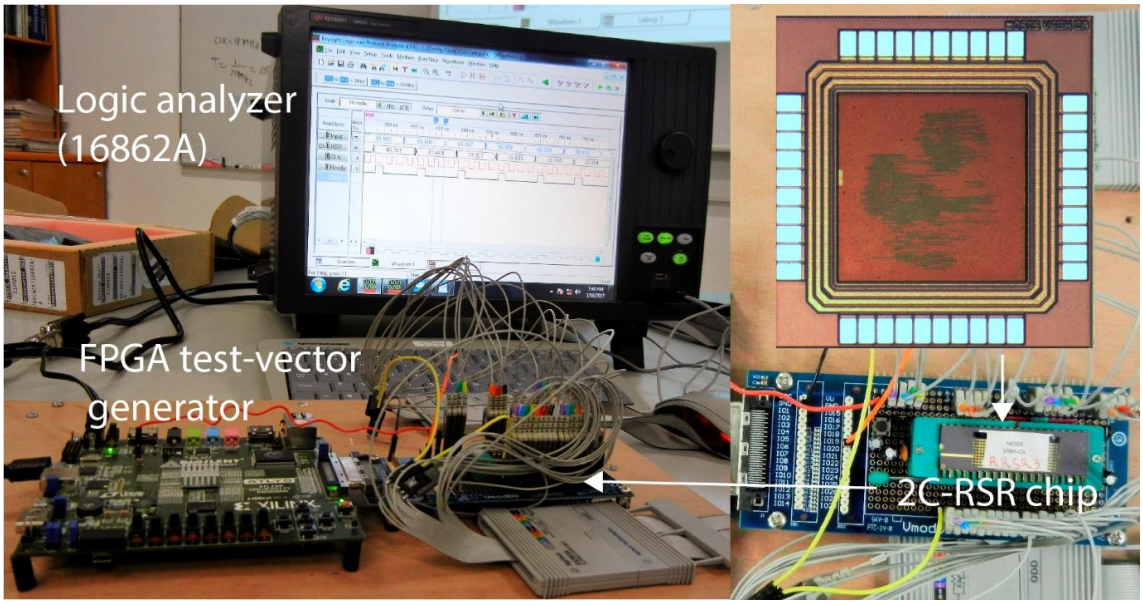

Fig. 3.6 Measurement of the 2C-RSR chip: test bench setup.

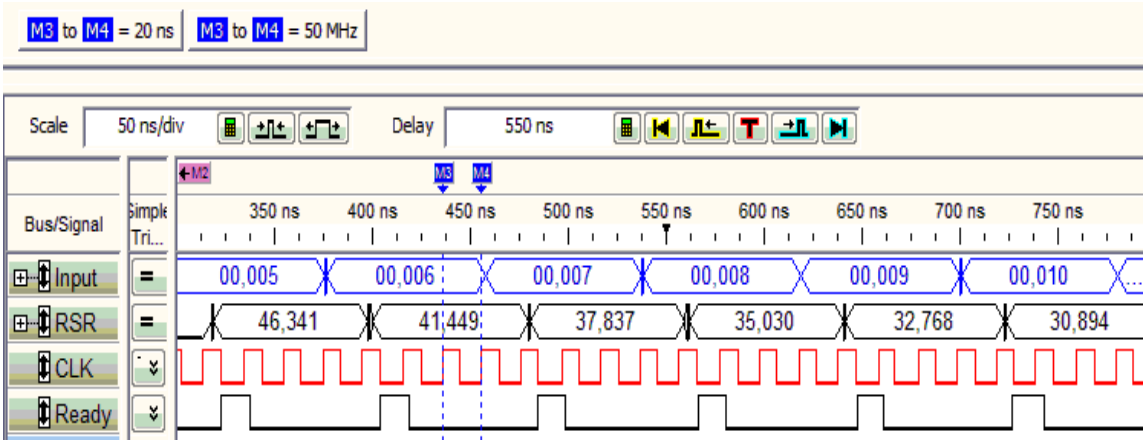

Fig. 3.7 Measurement of the 2C-RSR chip: timing diagram of the chip in operation.

3.4.2 Results of Chip Measurements

The timing diagram of the 2C-RSR-circuit functionality is shown in Fig. 3.7. The Input signal is the 16-bit *x* input to the chip, for which the RSR operation is computed. The 16-bit chip results are shown by the signal RSR represented in decimal format. To interpret these as $Q(16,11,u)$ FxP numbers, the decimal values should be multiplied by 2^{-11} (one ulp). For example, the Input = 00,008 multiplied by 2^{-11} yields 2^{-8}, and the corresponding output RSR = 32,768 yields 16, i.e., $1/\sqrt{2^{-8}} = 16$, which verifies that the chip result is correct for this value. The next signal in the timing diagram is the clock CLK; the cursors M3 and M4 show a rough measurement of the clock period, which corresponds to a frequency of 50 MHz. The Ready signal indicates that a new result is available. Note that the Ready signal remains at zero for three clock cycles; this is because the architecture requires two cycles for the RSR calculation and one cycle to put the result at the chip-output pins, which is not required when the design is interconnected with other units.

An exhaustive test of the 2C-RSR chip is performed applying all the possible input values to the chip and capturing its response using a logic analyzer. The 65,536 input-output data pairs are post-processed in Matlab to compare the chip output versus the golden values, which are calculated using the RSR operation in a double-precision floating-point (FP) computer and representing the results in $Q(16,11,u)$ format. From this comparison, we observe that all the chip results are exact for a maximum clock frequency of 49.62 MHz, i.e., the 65,536 RSR results are

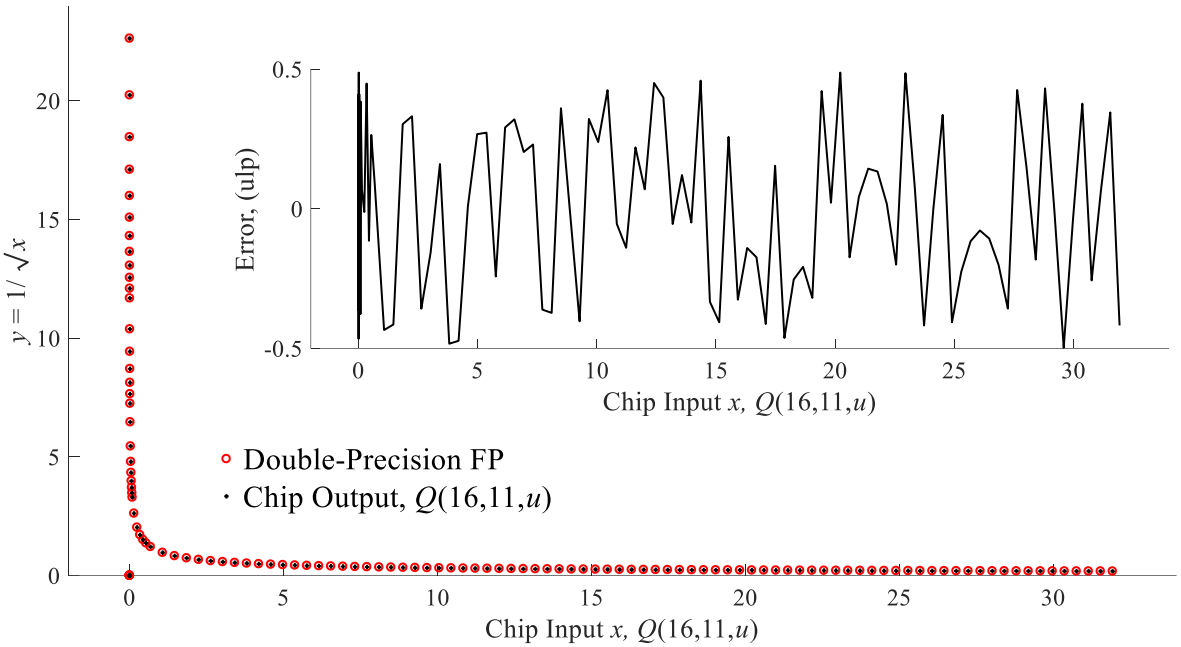

Fig. 3.8 2C-RSR-chip output and error versus the double-precision values.

bit-accurate with respect to the golden values. Additionally, in order to determine the accuracy of the 2C-RSR chip results versus those calculated by 64-bit computers, the chip results are compared with the corresponding double-precision FP values. The comparison is illustrated in Fig. 3.8; this shows that the maximum absolute error is ½ ulp of the $Q(16,11,u)$ format, i.e., the maximum absolute error of the 2C-RSR chip is 2^{-12} with respect to the values computed by a double-precision FP computer.

The detailed functionality of the 2C-RSR chip can be verified in Fig. 3.9, which shows the chip's response to the x input represented in radix 10 for the range of (0, 4]. The input and output data of the chip are acquired with the logic analyzer and then plotted. The labels show selected values, to illustrate the correct operation of the chip.

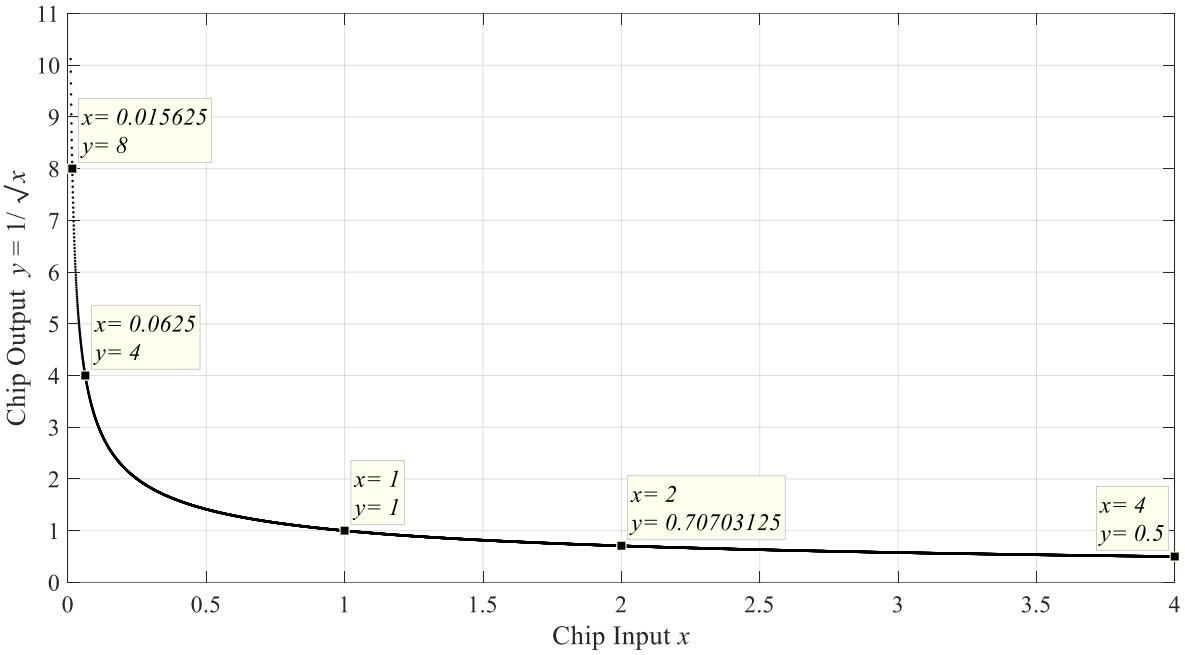

Fig. 3.9 2C-RSR chip-operating graph based on real data acquisition: input-output response.

The current consumption of the chip is measured by using a digital multimeter (Keysight U1241B) for several clock frequencies. The chip-core power supply (V_{DD}) is 1.2V. Using the current and voltage measurements, we calculate the total power consumption of the chip core. The core-total-chip power consumption for different clock frequencies is shown in Fig. 3.10. As can be seen, the power consumption is maintained approximately constant: $\Delta P/\Delta F = 0.1615$ mW/MHz, from 14 MHz to 44.21 MHz. Here, ΔP and ΔF denote increments of power consumption and frequency, respectively. From 44.21 MHz, $\Delta P/\Delta F$ decreases due to the reduction of swing voltage (V_{swing}) in internal nodes of the 2C-RSR integrated circuit, i.e., the dynamic power consumption follows the expression

$$P_{dynamic} = CFV_{DD}V_{swing} \tag{3-14}$$

where C is the average capacitance of the internal nodes, and F is the switching frequency. The reduction of V_{swing} is due to the inherent resistive-capacitive (RC) switching delay of the internal nets, which impose a maximum working frequency.

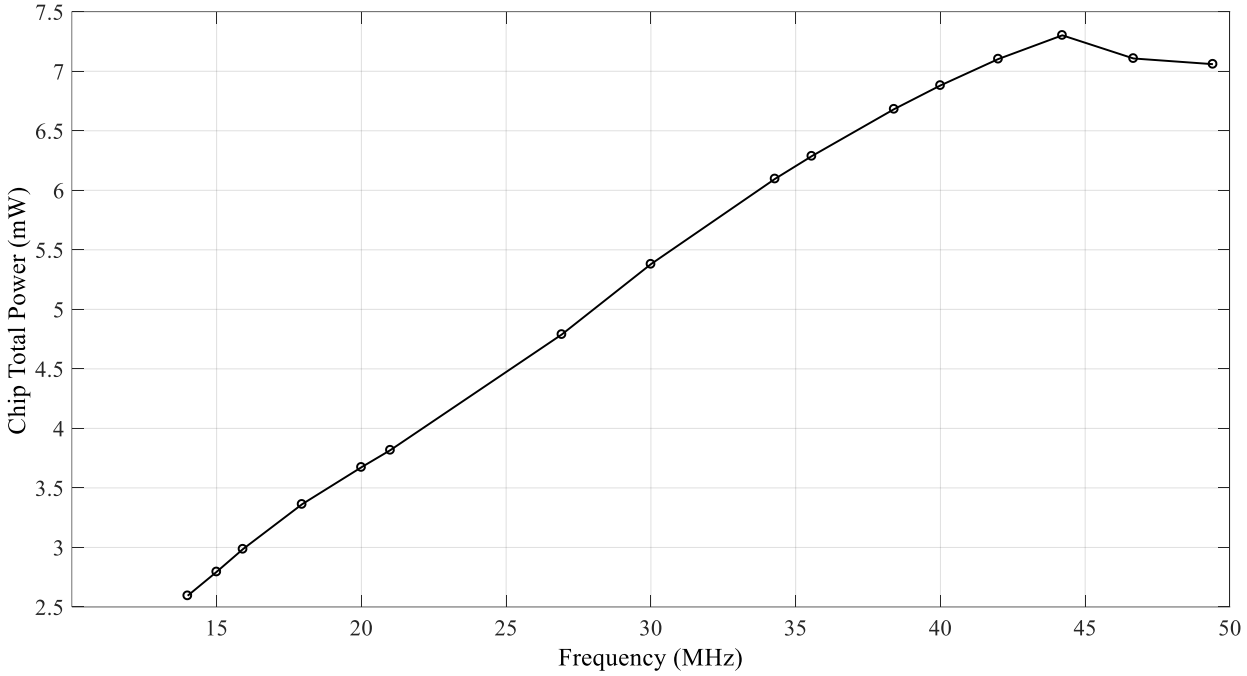

Fig. 3.10 Core total power consumption of the 2C-RSR chip versus clock frequency.

3.5. Comparison of 2C-RSR Chip with Previously Synthesized Designs

To the best of our knowledge, performance and results of on-chip implementations for RSR circuits have not been reported in the open literature. For comparison purposes, we use the simulation results of the ASIC designs presented in [Wires-06], [Kwon-08], and [Suresh-13], which are implemented at the level of logic synthesis using CMOS standard cells. The comparison outcomes are given in Table 3.5. As these designs are implemented using different feature-size technologies, Table 3.5 reports the original values of the comparison variables and the scaled values, which are calculated assuming Dennard scaling [Weste-11], i.e., the comparison variables of area, maximum working frequency, and power consumption of the three designs are scaled into the 130 nm feature size. The design in [Wires-06] and the 2C-RSR chip perform $1/\sqrt{x}$ directly, whereas the design in [Kwon-08] computes both $\sqrt{x}$ and $1/x$ operations. In the reported throughput values for computing $1/\sqrt{x}$ with the implementation of [Kwon-08], we consider the

TABLE 3.5. IMPLEMENTATION RESULTS COMPARED WITH REFERENCE DESIGNS

TS: transistor scaling; ClkF: clock frequency; CTP: consecutive throughput; NCTP: nonconsecutive throughput

Variable	[Wires-06]	[Kwon-08]	[Suresh-13]	This work
Operation	$1/\sqrt{x}$	$1/\sqrt{x}$	$\sqrt{x}$	$1/\sqrt{x}$
Word format	32 FP	32 FP	32 FP	16 FxP
On-chip realization	No	No	No	Yes
Feature size (nm)	250	90	65/40	130
Latency (cycles)	5	24	40	2
Area (mm^2)	0.4094	0.0453	0.0147/0.0057	0.2289
ClkF (MHz)	152.67	500	900	49.62
Power (mW)	92.87	NA	18.86/7.18	7.06
TS	0.52	1.44	2/3.25	1
Area×TS^2 (mm^2)	0.1107	0.0945	0.0588/0.0621	0.2289
ClkF/TS (MHz)	293.6	346.2	450	49.62
Power×TS^2 (mW)	25.11	NA	75.44/75.84	7.06
CTP (MOPS)	293.6	34.62	11.25	24.81
NCTP @ 100 MHz	20	4.17	2.5	24.81@ 50MHz

concatenation of $\sqrt{x}$ and $1/x$ operations. This implies 24 latency cycles. The design in [Suresh-13] only computes $\sqrt{x}$; hence, for computing $1/\sqrt{x}$ an extra division is required. It is important to note that $\sqrt{x}$ operation can be computed with the proposed 2C-RSR chip using only an additional multiplication, i.e., $\sqrt{x} = \left(1/\sqrt{x}\right)x$.

From Table 3.5, the design in [Suresh-13] has the smallest area and the highest clock frequency. In contrast, this design has the worst latency and throughput. The implementation with the lowest latency is the 2C-RSR chip, with only two clock cycles; this contributes to higher throughput for lower clock frequency. Furthermore, lower clock frequency means less dynamic power consumption and better signal integrity. For example, using the scaled values of Table 3.5, design in [Suresh-13] has a consecutive throughput (CTP) (i.e., when consecutive operations are being processed) of 11.25 million operations per second (MOPS) at a clock frequency of 450 MHz,

and a power consumption of 75.44 mW, whereas the 2C-RSR chip has a throughput of 24.81 MOPS at a clock frequency of 49.62 MHz and 7.06 mW. Hence, the 2C-RSR circuit doubled the throughput of design in [Suresh-13] with only 10% of power consumption and a clock frequency nine times lower.

Both designs presented in [Wires-06] and [Kwon-08] have better throughput than the 2C-RSR chip, at the expense of higher clock frequency and the negative effects on power consumption. The estimated power consumption of the design reported in [Wires-06] is 3.56 times higher than the proposed chip implementation. Designs presented in [Wires-06] and [Suresh-13] are attractive for high-performance single-precision FP-units, in applications where power constraint is not important; nevertheless, mobile-embedded systems require power consumption reduction and high-clock frequencies are not desired due to the dynamic-power consumption of CMOS technology.

Current commercial low-power embedded microcontrollers work in the range of few tens of MHz. For example, the NXP microcontrollers L-Series and K-Series based on ARM Cortex-M4 operate in the range of 48-96 MHz and 50-180 MHz, respectively[41]. For this reason, the last row of Table I shows the nonconsecutive throughput (NCTP) comparison of all reference designs running with a practical-clock frequency of 100 MHz; since the 2C-RSR is unable to run at 100 MHz, its maximum throughput is used. The results confirm the low-latency advantage of the 2C-RSR chip, which performs the fastest RSR computation for nonconsecutive operations.

3.6. Conclusions

The on-chip implementation of a fixed-point RSR unit has been presented. Its main characteristics are low latency and bit-accurate results for the $Q(16,11,u)$ format. Even though the RSR operation is widely used in mobile-multimedia devices, to the best of our knowledge, its FxP on-chip implementation has not been reported before. The architecture of the presented digital integrated circuit is based on the Newton-Raphson method, where the seed is provided by a piecewise-polynomial approximation in a reduced range of the RSR function. The 2C-RSR chip

[41] NXP Processors and Microcontrollers, Low-power 32-bit Microcontrollers. Nov. 11, 2017, https://www.nxp.com/products/processors-and-microcontrollers/arm-based-processors-and-mcus/kinetis-cortex-m-mcus:KINETIS.

produces a new result in only two clock cycles, with a maximum error of 2^{-12}, which is equivalent to ½ ulp of the $Q(16,11,u)$ format.

The chip was manufactured using a 130 nm CMOS process; the standard-cell area is 0.2289 mm^2 and the total chip area including the 40-I/O-pad frame is 1.598 mm^2. The measured power consumption is 7.06 mW at the maximum clock frequency of 49.62 MHz. Comparisons of the 2C-RSR-chip measurements with simulation results of existing standard-cell-based implementations show that the proposed implementation exhibits the lowest latency. Furthermore, the 2C-RSR-chip power consumption is one order of magnitude lower than the design reported in [Suresh-13], and 3.56 times lower than the estimated in [Wires-06]. This makes the presented design suitable as a silicon intellectual property for applications with low latency and low-power consumption in mobile devices with severe hardware constraints.

4. IEEE-754 Half-Precision Floating-Point Low-Latency RSR IP-Core

The IEEE 754-2008 standard for floating-point (FP) arithmetic defines a binary interchange format of 16 bits, i.e., binary16, which can be used for the exchange of FP data between different implementations [IEEE-08]. The binary16 format, better known as half-precision floating-point (FP16) format, has been widely applied in gaming and other applications. For example, a programmable renderer is reported in [Peercy-00], where it is demonstrated that the FP16 format is sufficient for most of the shaders performed. In addition, interest in FP16 arithmetic has arisen because in the breakthrough neural-network technique known as deep learning, most of the math required to train neural networks can be executed in FP16 arithmetic [Foley-17]. For instance, a convolutional neural-network accelerator has been implemented using FP16 arithmetic [Venkatesh-17], achieving high accuracy and performance in image classification, while reducing computational requirements. These works show the advantages of the FP16 format for low-precision tolerant applications.

In the arena of approximate-computing applications, FP16 arithmetic is becoming relevant for designs that require low-power consumption and low computational cost [Yin-16]. In fact, NVIDIA recently added native FP16 computational support to some of its GPU architectures in order to take advantage of FP16 performance for deep-learning applications[42] [Foley-17]. Moreover, Intel has added instructions to convert FP16 values to/from single-precision floating-point (FP32) numbers[43]. The main advantages of FP16 format over the FP32 format are: adequate accuracy for many applications, half the storage space requirements, half the memory bandwidth, and better speed performance.

This chapter describes an FPGA implementation of the 2C-RSR algorithm that is presented in Section 3.2 of this thesis document, with the variation that the new implementation is performed by using half-precision floating-point arithmetic. The new implementation is named HF-2cRSR.

Several 64-bit floating-point (FP) RSR architectures have been developed [Lang-03],

[42] Anandtech, The NVIDIA GeForce GTX 1080 & GTX 1070. Jun. 17, 2017, http://www.anandtech.com/show/10325/the-nvidia-geforce-gtx-1080-and-1070-founders-edition-review.
[43] Intel Software, Overview: Intrinsics to Convert Half Float Types. Jun. 21, 2017, https://software.intel.com/en-us/node/524286.

b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
Sign	Exponent E					Significand m									
S	E_4	E_3	E_2	E_1	E_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}

Fig. 4.1 IEEE 754-2008 half-precision floating-point format.

[Piñeiro-02], however, since they are mainly focused on accelerating high-precision FP units, they are not suitable for embedded systems. Similarly, 32-bit FP RSR designs have been reported [Suresh-13], [Butts-11]. Although 32-bit FP architectures can be applied in embedded mobile devices, their power consumption and implementation area make them poorly suited for low-power embedded applications that do not require high accuracy. For mobile low-power systems, fixed-point (FxP) implementations are preferred due to their reduced hardware area and better performance [Wang-10], [Aguilera-Galicia-18], [Liu-17a], however, the dynamic range of FxP formats are smaller than FP systems with the same number of bits, and their interface with FP systems requires additional resources.

The HF-2cRSR is implemented on two FPGA technologies and the results are compared with similar intellectual-property (IP) cores [Xilinx-12], [Altera-16] of the top-two FPGA companies: Xilinx and Intel. The results show that the proposed half-precision implementation exhibits better throughput than Intel IP with 30% resource savings. With respect to the Xilinx IP, the HF-2cRSR gives 66% more throughput at the cost of more LUT resources. In both comparisons, the proposed implementation exhibits lower latency and better throughput at lower clock frequency. All the above makes the proposed IP core suitable for low-cost, half-precision embedded applications where area and power consumption are important design trade-offs.

4.1. Introduction to Half-Precision Floating Point Numbers

The FP16 format is shown in Fig. 4.1. The representation is based on three fields: sign, *S*; biased exponent, *E*; and trailing significand, *m*.

All FP numbers, including zeros and infinities, are signed. The exponent is encoded using an offset-binary representation, i.e., a constant that is known as *bias* is added to the exponent to make the biased-exponent range non-negative. For the FP16 format, the exponent bias is 15_{10}. When $E \neq 0$, the FP number is of the normalized type and its value, x_N, is computed using

TABLE 4.1. SUMMARY OF HALF-PRECISION FLOATING-POINT ENCODING

Decimal Exponent E	Significand $m = 0$	Significand $m \neq 0$	Equation
0	+/- 0	Denormalized	$x_D = (-1)^S\, 0.m \times 2^{-14}$
1 – 30	Normalized		$x_N = (-1)^S\, 1.m \times 2^{E-15}$
31	+/- ∞	Not-a-Number	-

$$x_N = (-1)^S\, 1.m \times 2^{E-bias} \tag{4-1}$$

where E is defined by

$$E = \sum_{i\geq 0}^{i=4} E_i \times 2^i, \tag{4-2}$$

and m is defined by

$$m = \sum_{i\geq 1}^{i=10} m_i \times 2^{-i}. \tag{4-3}$$

For normalized FP numbers, it is assumed that there is an implicit-leading bit of value one in the significand, which have an 11-bit precision (only ten bits of the significand are stored in memory).

When the exponent is zero (E = 0), the FP value is zero (+/- 0) if the significand is zero; otherwise, the FP number is of the denormalized type. The value of a denormalized FP16 number is computed using

$$x_D = (-1)^S\, 0.m \times 2^{-14}. \tag{4-4}$$

When the exponent is 31_{10}, the FP16 value is infinite (+/- ∞) if the significand is zero; otherwise, the FP16 value is not-a-number (NaN). Table 4.1 shows a summary of the FP16 encoding depending on the exponent and significand values.

4.2. Half-Precision Floating-Point RSR Operation

The proposed half-precision FP RSR IP core computes the operation defined by (3-1), where x, and y in this case are binary half-precision FP numbers, which can be normalized or denormalized according to [IEEE-08]. When x is positive normalized or denormalized, the

TABLE 4.2. OUTPUT RESULTS FOR THE HF-2CRSR IP CORE
qNaN: quiet not-a-number

	x input				
	+0	negative	Denormalized / Normalized	+∞	NaN
y output	+∞	qNaN	$\circ\left(1/\sqrt{x}\right)$	+0	qNaN

correctly rounded value $\circ\left(1/\sqrt{x}\right)$ must be given back as a half-precision FP number. For other special values of *x*, the RSR operation must return the results that are shown in Table 4.2. The special value quiet not-a-number, qNaN, is used to represent the result of an invalid operation exception without signaling exceptions [Muller-10].

4.3. HF-2cRSR Architecture

The proposed IP-core architecture is based on the algorithm introduced in Section 3.2. As is explained in such section, this algorithm applies the NR method. The required seed is computed using a piecewise-polynomial approximation in a reduced range, *rr* = [0.5, 2) of the RSR function, which contributes to provide a seed with enough accuracy to achieve the IEEE 754-2008 half-precision accuracy standard in only one NR iteration. The proposed hardware FP architecture is represented in Fig. 4.2. In the following sections, the design considerations to implement the corresponding blocks of the HF-2cRSR architecture using floating-point arithmetic, are described.

4.3.1 Floating-Point Scaling and De-scaling Operations

The scaling and de-scaling operations are performed implementing the products defined by (3-6), and (3-11) respectively. These operations are represented by the Scaler and De-scaler blocks in Fig. 4.2.

To implement the scaling operation, it must be detected if the input *x* is of the normalized or denormalized type, that is why the Scaler module requires the normalized *x* flag, Norm, as shown in Fig. 4.2. When the input *x* is of the denormalized type, it must be normalized by left shifting the

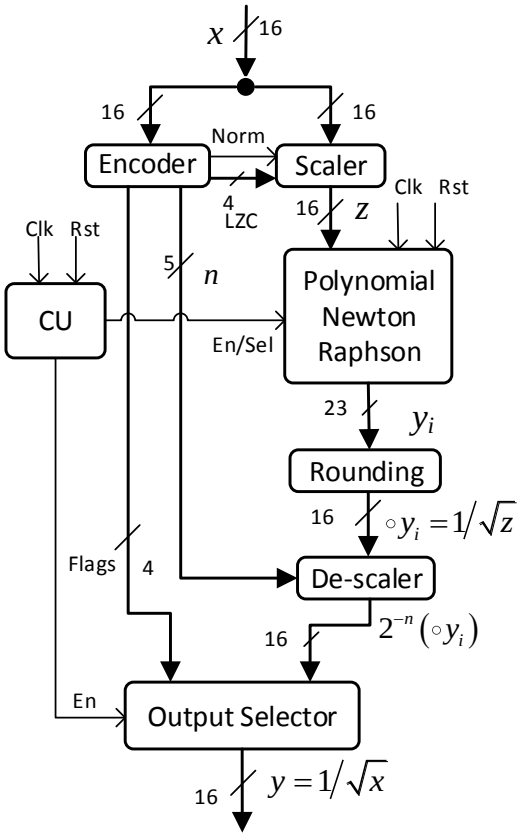

Fig. 4.2 Architecture of the half-precision floating-point RSR, HF-2cRSR.

significand bits by the number of leader zeros of the significand, LZC, plus one. The Norm flag and LZC count are provided by the Encoder module that is presented in Section 4.3.4. Considering that in the proposed design, the possible *z* exponent values are only 0 or -1 since $0.5 \le z < 2.0$, the implementation of the scaling operation can be simplified as follows. For normalized inputs, the scaled exponent is determined by evaluating only the least significant bit (LSB) of the *x* exponent: if LSB of *x* is zero, then z exponent = -1, else 0. For denormalized inputs, the scaled exponent is determined by checking the LSB of the LZC count: if LSB is zero, then z exponent = 0, else -1. A multiplexer is used to select the correct value of the z exponent. For this reason, the scaling exponent, *n*, is only connected to the De-scaler module.

4.3.2 Polynomial and Newton-Raphson Method

To evaluate the Newton-Raphson iteration for the RSR function defined by (3-7), the

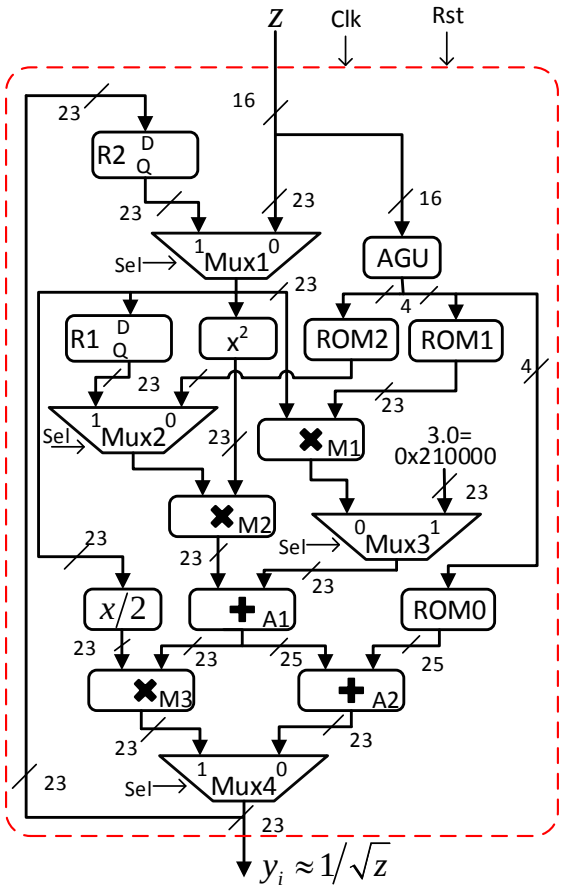

Fig. 4.3 Floating-point polynomial Newton-Raphson architecture of the HF-2cRSR.

following FP-arithmetic blocks are required: one squaring module, two multipliers, one adder, and one divider by two.

In the proposed architecture, the seed y_0 is computed using a piecewise-polynomial approximation in the *rr* range. The polynomial fitting is performed by splitting the *rr* range into 14 unequal subintervals, as is explained in Section 3.3.3; each subinterval is approximated by a second-order polynomial defined by

$$y_0 = a_0 + a_1 z + a_2 z^2 \tag{4-5}$$

where a_i represent the *i*-th polynomial coefficient. To evaluate (4-5), the following FP-arithmetic blocks are required: one squaring module, two multipliers, and two adders. Since some of these blocks can be reused, (3-7) and (4-5) are performed in the Polynomial Newton-Raphson (PNR) module of Fig. 4.2. The detailed architecture of the PNR module is shown in Fig. 4.3. This module

requires two clock cycles to compute an approximation of $1/\sqrt{z}$: in the first clock cycle it evaluates (4-5) and in the second cycle it computes one iteration of (3-7).

To calculate the seed using (4-5), the 14 polynomial coefficients a_0, a_1, and a_2 are stored in read only memories ROM0, ROM1, and ROM2, respectively (see Fig. 4.3). The address generator unit AGU provides the ROM address of the corresponding coefficients for the z subinterval. The z^2 term is computed in the x^2 squaring module. The products a_1z, and a_2z^2 are performed in the multipliers M1 and M2, and the sums are computed in the adders A1 and A2, respectively. In order to achieve the required accuracy for the HF-2cRSR unit, the FP arithmetic blocks (x^2, M1, M2, A1, and A2) are word size customized. The multipliers use 23-bit word length: 1-bit sign, 5-bit exponent, and 17-bit significand (18 bits of precision, considering the implicit-leading bit); the products are rounded to the nearest FP value, with round-to-nearest-even as the tie-breaking rule. The design and architecture of the 23-bit FP multiplier are reported in Section 4.4. The adders use 25-bit word length: 1-bit sign, 5-bit exponent, and 19-bit significand. The significand sizes of the FP blocks are determined by circuit-level simulations to achieve the maximum allowed error defined by the IEEE-754 standard for the RSR operation, which is ½ unit in the last place (ulp). The multipliers and adders are modeled by portable and structured Verilog models that implement canonical architectures for these operations [Muller-10].

For the computation of the NR iteration (3-7), the zy_i^2 term is performed in the x^2 squaring block and the M2 multiplier; the $3\text{-}zy_i^2$ difference is realized in the A1 adder; and the $(y_i/2)(3\text{-}zy_i^2)$ product in the M3 multiplier. As can be seen, the proposed architecture reuses some FP blocks to reduce hardware resources. The $y_i/2$ operation is efficiently implemented by decrementing the exponent by one, since the PNR module is working with normalized numbers. This is another advantage of the proposed architecture: since the approximation is performed in the *rr* range, all the computations inside the PNR module use normalized numbers, which contributes to fewer hardware resources in the FP arithmetic blocks.

4.3.3 Floating-Point Rounding Operation

The output of the PNR module has the customized 23-bit FP format. To obtain the half-precision format, the output y_i is rounded to 16 bits. This operation is implemented in the Rounding

TABLE 4.3. INTERVALS OF *x* AND SCALING EXPONENTS FOR DENORMALIZED NUMBERS

Hexadecimal Mantissa	FP Low Boundary $\times 10^{-4}$	FP Up Boundary $\times 10^{-4}$	Scaling Exponent *n*
000 - 001	0	0.0005960	–12
002 - 007	0.0011921	0.0041723	–1
008 - 01F	0.0047684	0.0089407	–10
020 - 07F	0.0190735	0.0756979	–9
080 - 1FF	0.0762939	0.3045797	–8
200 - 3FF	0.3051758	0.6097555	–7

block that is shown in Fig. 4.2, where a round-to-nearest operation is performed. Once the approximation to $1/\sqrt{z}$ is rounded, $\circ y_i$, this value is de-scaled to obtain the half-precision result with an error $\leq$ ½ ulp.

4.3.4 Encoder Module

In order to apply the scaling operation, the *x* input range that includes denormalized and normalized values is divided into 21 intervals. One scaling factor is associated to each interval, in this way, all the intervals of *x* can be scaled into the *rr* range. The scaling exponents for the denormalized and normalized *x* values are reported in Table 4.3 and Table 4.4, respectively. The function of the Encoder module is to identify the corresponding interval of the *x* input value and provide the corresponding scaling exponent *n* to the De-scaling block. When $x < 0.5$ (below the *rr* range), *n* is negative. The implementation of the scaling operation is different for normalized *x* values than for denormalized values of *x*. For this reason, the Encoder generates the Norm flag to indicate to the Scaler block when the input *x* is of the normalized type. When the input *x* is of the denormalized type, it must be first normalized; to normalize *x*, the Encoder module provides the leading zeros count, LZC, of *x* to the Scaler module. Furthermore, the Encoder module generates the flags negative input, zero, infinite, and NaN, which are useful to determine the output result and signaling these special values.

TABLE 4.4. INTERVALS OF *x* AND SCALING EXPONENTS FOR NORMALIZED NUMBERS

Hex. Biased Exponents	Dec. Unbiased Exponents	FP Low Boundary	FP Up Boundary	Scaling Exponent *n*
01	–14	0.0000610	0.0001220	–7
02, 03	–13, –12	0.0001221	0.0004880	–6
04, 05	–11, –10	0.0004883	0.0019522	–5
06, 07	–9, –8	0.0019531	0.0078087	–4
08, 09	–7, –6	0.0078125	0.0312347	–3
0A, 0B	–5, –4	0.03125	0.1249390	–2
0C, 0D	–3, –2	0.125	0.4997559	–1
0E, 0F	–1, 0	0.5	1.99902344	0
10, 11	1, 2	2	7.99609375	1
12, 13	3, 4	8	31.984375	2
14, 15	5, 6	32	127.9375	3
16, 17	7, 8	128	511.75	4
18, 19	9, 10	512	2047	5
1A, 1B	11, 12	2048	8188	6
1C, 1D	13, 14	8192	32752	7
1E	15	32768	65504	8

4.3.5 Output Selector and Control Unit

The results of the RSR operation have special values that are computed out of the general data-path. The Output Selector module selects the correct result depending on the *x* input value and the flags provided by the Encoder block. The possible output values of the HF-2cRSR are shown in Table 4.2.

The control unit CU is a finite-state machine that synchronizes the operation of the PNR block and enables an output register when a new result is available.

4.4. Floating-Point Multiplier for the HF-2cRSR IP-Core

The design and implementation of a customized floating-point multiplier is presented in

this section. The proposed multiplier is one of the fundamental modules of the HF-2cRSR IP-core. Some of the FP multiplier specifications are customized for the HF-2cRSR, for example, the word size, processing of normalized/denormalized numbers, and rounding mode are selected to reduce the hardware implementation of the multiplier and to achieve a bit-accurate half-precision FP RSR unit. The verification of the FP multiplier is presented and its logic synthesis is performed on ARM 130 nm CMOS ASIC technology. The results of the logic synthesis, such as required standard-cells, silicon area, operating frequency, and power consumption, are reported.

4.4.1 Design Requirements of Floating-Point Multiplier

This section defines the design requirements of a FP multiplier for the HF-2cRSR unit. Specifications such as the word size, input-output range, and rounding mode of the FP multiplier are defined.

4.4.1.1 Input and Output Range of the FP Multiplier

The input to the PRN module of the HF-2cRSR is in the reduced range, $rr = [0.5, 2)$. For this reason, the specification of the input range for the FP multiplier can be reduced to handle only normalized FP numbers. Furthermore, since the PNR module computes an approximation of the RSR operation in the *rr* range, the PNR output range is $(1/\sqrt{2}, \sqrt{2}]$. As the output of the PNR module is bounded, the output range of the multiplier is bounded; this means that the number of bits required to represent the exponent in the FP multiplier can be reduced. The selected word size for the FP multiplier is discussed in the following section.

4.4.1.2 Word Size of the FP Multiplier

The representation range of an FP format is mainly determined by the number of bits, *w*, of the exponent, whereas the precision of an FP format is determined by the number of bits, *p*, of the significand. In the following paragraphs, the *w* and *p* values for the FP multiplier word size are defined.

The HF-2cRSR unit must provide a result in FP16 format, which has a representation range of normalized FP numbers of $2^{-14} \le |x_N| \le 65,504$. This range is greater than the expected for the

multiplier products inside the PNR module because the output of the later is bounded. The above means that five bits are enough for the multiplier exponent, i.e., $w = 5$. Although w can be reduced, in order to work with an exponent size compatible with the FP16 standard, we selected five bits to represent the exponent of the FP multiplier.

The number of bits, p, of the significand, define the precision of the FP multiplier and the size of the internal product of the significands, $2p$ bits, which determines the amount of hardware resources of the fixed-point multipliers required to multiply the significands. For this reason, it is important to select the multiplier significand size as small as possible without affecting the required accuracy in the HF-2cRSR unit. The accuracy goal for the HF-2cRSR unit is ½ ulp of the FP16 format, i.e., a maximum error of 2^{-11}. By circuital simulations of an RSR preliminary model, showed that a multiplier significand size of at least 18 bits is required to achieve the accuracy goal.

From the above, the FP multiplier word size is 23 bits: one sign bit, five bits for the exponent, and a 17-bit significand. The implicit-leading bit is not part of the word size.

4.4.1.3 Rounding Mode of the FP Multiplier

The IEEE 754-2008 standard defines that an FP multiplier product must be rounded, and establishes four rounding modes [IEEE-08], [Muller-10]. The recommended default mode is the round-to-nearest, with round-to-nearest-even as the tie-breaking rule. This rounding mode is presented in the following paragraphs.

The result of rounding a normalized FP product, Z, using p bits of precision is either: a) the FP number

$$Z_p = (-1)^{S_Z} \left(m_0.m_1m_2...m_{p-1}\right)_2 2^{e_Z} \tag{4-6}$$

obtained by truncating the significand m after p-1 bits; or b) the *successor* of Z_p, which is obtained by

$$Z_{ps} = Z_p + 2^{-p+1}. \tag{4-7}$$

The choice between (4-6) or (4-7) depends on the rounding bit, m_p, and the sticky bit, which is the bit that says there is at least one non-zero bit among the remaining bits to the right of m_p. The sticky bit may be evaluated by the logical *OR* operation of all the bits to the right of m_p, $OR\left(m_{p+1}, m_{p+2}, ...\right)$. A summary of the criteria used for rounding the FP multiplier product is shown

TABLE 4.5. ROUNDING TO NEAREST WITH TIE TO EVEN CRITERIA

Round Bit m_p	Sticky Bit $OR(m_{p+1}, m_{p+2}\ldots)$	Rounded Number
0	0	Z_p
0	1	Z_p
1	0	Even of $\{Z_p, successor(Z_p)\}$
1	1	$successor\ (Z_p)$

TABLE 4.6. SPECIFICATION OF THE 23-BIT FP MULTIPLIER

Variable	Value	Comment
Input range	[0.5, 2)	Normalized FP number
Minimum output range	$(1/\sqrt{2}, \sqrt{2}]$	Normalized FP number
Exponent size	5 bits	Half-precision format compatible
Significand size	17 bits	Required size for the HF-2cRSR unit
Rounding mode	round-to-nearest	Tie-breaking rule: to-nearest-even

in Table 4.5.

The specifications of the 23-bit FP multiplier are summarized in Table 4.6.

4.4.2 23-Bit FP Multiplier Architecture

This section describes the multiplication specification for two FP numbers and the hardware architecture of a customized 23-bit FP multiplier for the HF-2cRSR unit.

4.4.2.1 Floating-Point Multiplication

The multiplication product, Z, of two nonzero binary FP numbers $X = (-1)^{S_X}\left(|X|\right)$ and $Y = (-1)^{S_Y}\left(|Y|\right)$ satisfies

$$Z = X \times Y = (-1)^{S_Z}\left(|X| \times |Y|\right) \tag{4-8}$$

TABLE 4.7. SPECIFICATION OF MULTIPLICATION FOR POSITIVE FP NUMBERS

$\|X\|\times\|Y\|$		$\|Y\|$			
		+0	*De/Normalized*	+∞	NaN
$\|X\|$	+0	+0	+0	qNaN	qNaN
	De/Normalized	+0	$\circ(\|X\|\times\|Y\|)$	+∞	qNaN
	+∞	qNaN	+∞	+∞	qNaN
	NaN	qNaN	qNaN	qNaN	qNaN

where

$$S_Z = S_X \oplus S_Y \in \{0,\ 1\} \tag{4-9}$$

is the sign of the product, $|Y| = m_Y 2^{e_Y}$, $|X| = m_X 2^{e_X}$, and

$$(|X|\times|Y|) = m_X m_Y \cdot 2^{e_X + e_Y} \tag{4-10}$$

where $e_X = E_X - bias$ and $e_Y = E_Y - bias$ represent the unbiased exponents of X and Y respectively. The IEEE 754-2008 specification for (4-10) is summarized in Table 4.7. The following section presents the FP multiplier architecture, for the case where both numbers X and Y are normalized.

4.4.2.2 Hardware Architecture of the Floating-Point Multiplier

To implement the multiplication of two normalized FP numbers, their three format fields– sign, exponent, and significand– are processed separately. The proposed hardware architecture of the 23-bit FP multiplier is shown in Fig. 4.4. This implements the previously defined equations for the multiplication operation. The sign of the product, S_Z, is implemented by using (4-9).

From (4-10), the exponent of the result is computed by

$$e_X + e_Y = E_X - bias + E_Y - bias \tag{4-11}$$

where E_X and E_Y are the biased exponent of X and Y respectively. As the FP format uses a biased exponent, the result biased exponent E_Z can be computed by

$$E_Z = E_X + E_Y - bias. \tag{4-12}$$

The computation of (4-12) is represented in Fig. 4.4 by the left branch of Mux1. Due to the

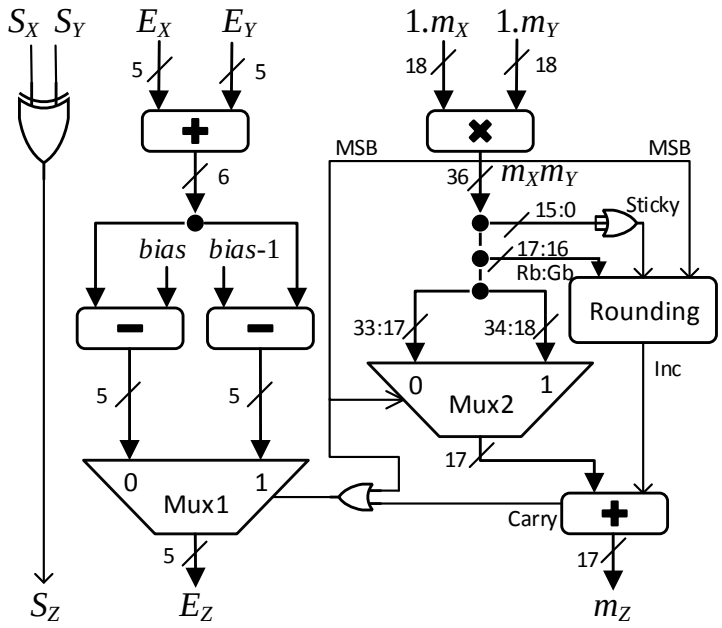

Fig. 4.4 Architecture of the 23-bit floating-point multiplier.

rounding operation, the computation of (4-7) can produce an overflow of the significand; this change in the significand value must be compensated by incrementing the exponent. This operation is selected by the right branch of Mux1.

The product $m_X \times m_Y$, represented by $m_X m_Y$ in Fig. 4.4, is performed using a fixed-point multiplier. Because the multiplier inputs are of the normalized type, i.e., $1 \le m_X < 2$ and $1 \le m_Y < 2$ the product of significands satisfices $1 \le m_X m_Y < 4$. As the multiplication result must be of the normalized type, when $m_X m_Y \ge 2.0$, it needs to be normalized by right-shifting $m_X m_Y$ one position when its most significant bit, (MSB), is one. This is the purpose of Mux2, which selects the normalized product, $m_X m_Y$ [33:17], or the shifted product, $m_X m_Y$ [34:18], depending on MSB.

The Rounding block of Fig. 4.4 is the control logic for the rounding operation as presented in Section 4.4.1.3. If MSB = 0, the guard bit, Gb in Fig. 4.4, is used as the rounding bit, and the sticky signal is used as the sticky bit. When MSB=1, the Rb signal is used as the rounding bit and the sticky bit is the result of the logic-*OR* operation between the signals Gb and sticky, *OR*(Gb, Sticky). The output adder of Fig. 4.4, performs the computation of (4-7), utilizing the signal Inc and the selected bits of $m_X m_Y$. The result of the operation *OR*(Carry, MSB) selects the correct

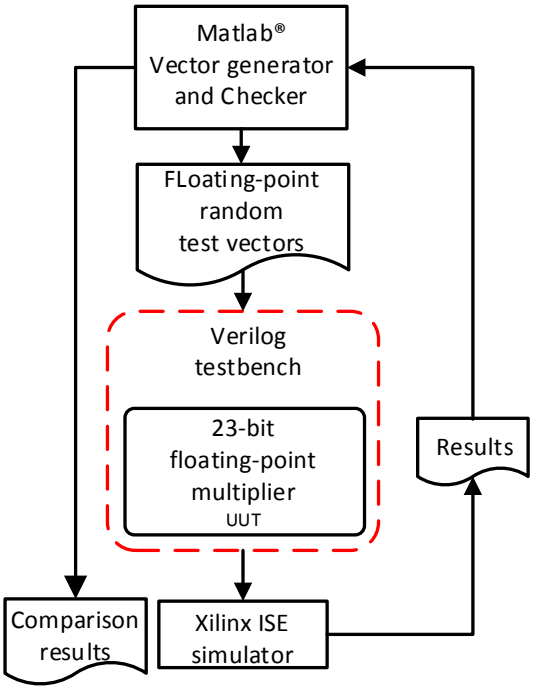

Fig. 4.5 Test environment for verifying the 23-bit floating-point multiplier.

exponent value. The final rounded result, Z, of the 23-bit FP multiplier is the concatenation of the signals $\{S_Z, E_Z, m_Z\}$ in Fig. 4.4.

4.4.3 Implementation and Verification of the 23-Bit FP Multiplier

The architecture of the 23-bit FP multiplier, presented in Section 4.4.2, is implemented by a hardware description language (HDL) model. The multiplier model is done using portable and technology independent Verilog code. This model describes a hierarchical structure based on mutiplexers, adders, subtractors and one fixed-point multiplier. The 23-bit FP multiplier model is verified using the test environment shown in Fig. 4.5. To determine the performance and the hardware resources required for implementation of the propose multiplier, the verified design is synthesized on 130 nm ASIC technology by using the Cadence® RTL-Compiler synthesis tool. The following sections report the functional verification of the proposed design and its synthesis results.

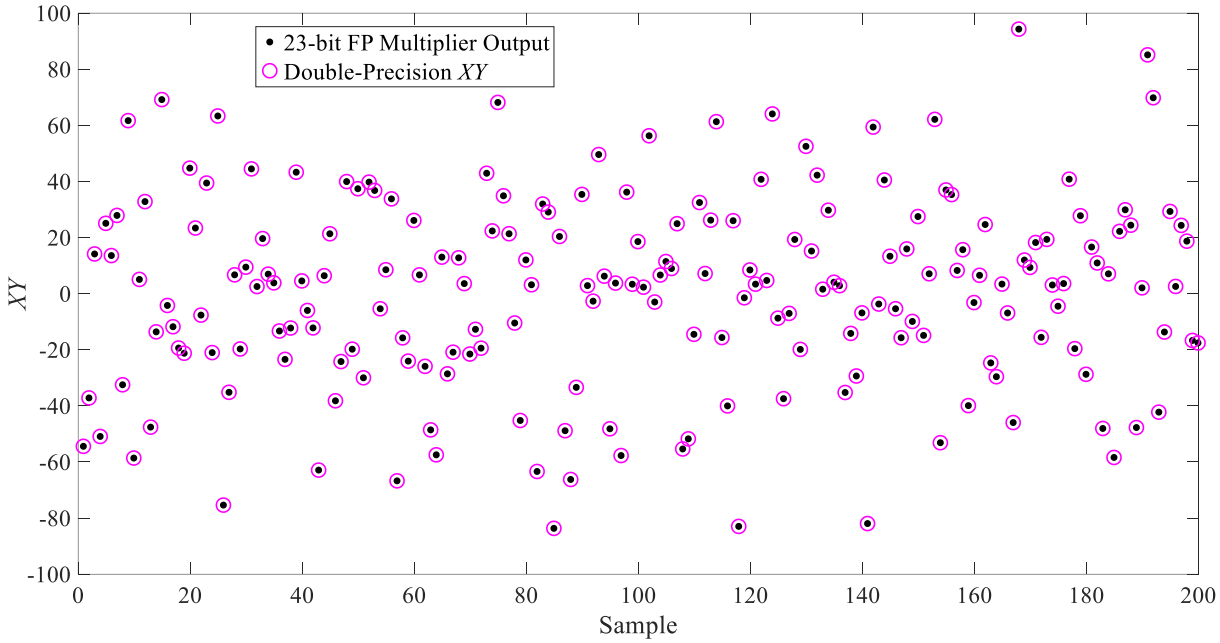

Fig. 4.6 Functional verification of the 23-bit floating-point multiplier and comparison of the output products with respect to the corresponding double-precision calculated values. Reduced-range random inputs ($-10 < X, Y < 10$) are applied to the multiplier.

4.4.3.1 Verification of the 23-bit FP Multiplier

The 23-bit FP multiplier has 46 inputs (23 for X and 23 for Y). To perform an exhaustive verification of this multiplier, 2^{46} test cases are required. To verify the proposed multiplier, a test-case subset is selected. This subset corresponds to the expected operating range of the multipliers inside the PNR module of the HF-2cRSR unit. Rough estimations show that the multiplier input range is [–10, 10].

The functional verification of the 23-bit FP multiplier is performed by using a Verilog testbench, which applies the test vectors to the unit under test, UUT, as seen in Fig. 4.5. Because the input-output test vectors have a nonstandard FP format, these are generated by using Matlab that helps to analyze and compare the multiplier simulation results. The generated test vectors are sequences of uniformly distributed pseudo-random 23-bit FP numbers.

The simulations are performed using the Xilinx® ISE simulator. The multiplier's products and the input test vectors are exported to a file for their processing and verification using Matlab, where the 23-bit results are compared with the corresponding double-precision calculated values.

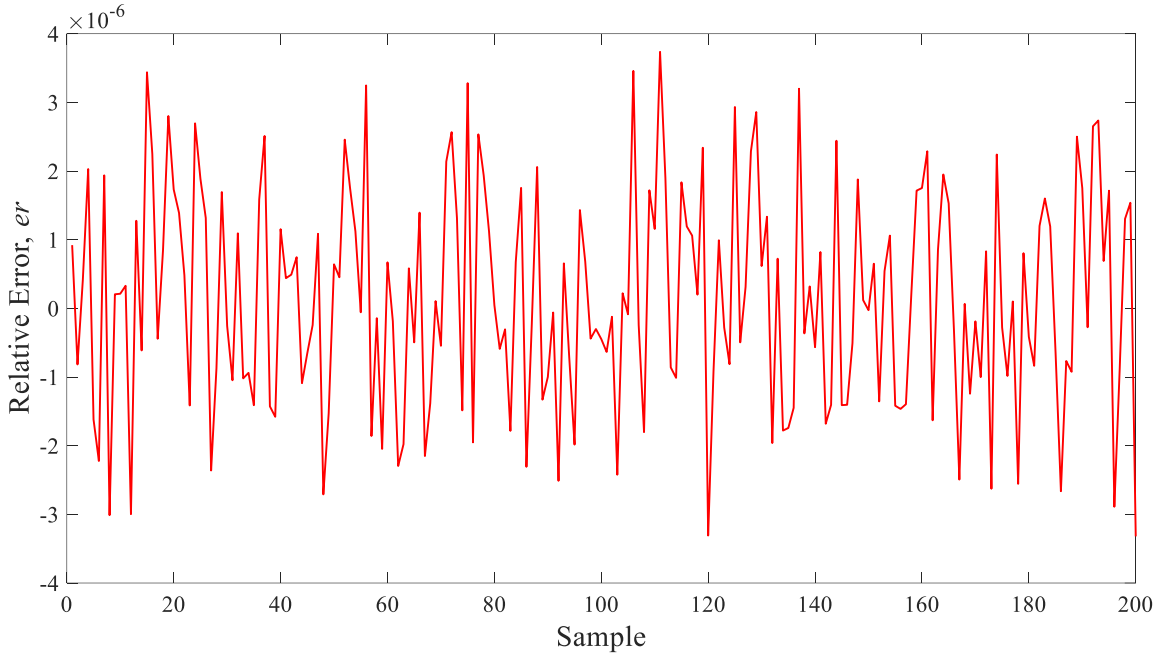

Fig. 4.7 Relative error of the 23-bit FP multiplier versus double-precision FP computation.

A demonstrative simulation is presented in Fig. 4.6. It shows 200 random multiplication products, represented by *XY*, which are compared with respect to the corresponding calculated values using a 64-bit computer. It can be seen that the 23-bit FP multiplier outputs are close to the corresponding double-precision values.

In order to know the error of the 23-bit FP multiplier with respect to the corresponding products computed by a 64-bit computer, Z_{DP}, the relative error is computed by using

$$e_r = \left| \frac{Z - Z_{DP}}{Z_{DP}} \right|. \tag{4-13}$$

The relative error of the 23-bit FP multiplier with respect to the double-precision FP computation is plotted in Fig. 4.7. It can be observed that for this demonstrative simulation the relative error is $-4\times10^{-6} < e_r < 4\times10^{-6}$. Simulations of the 23-bit FP multiplier with more test-case coverage are performed; a histogram of the relative errors of a simulation for one million test cases is shown in Fig. 4.8, where the error is reported in units-in-the-last-place (ulp) [Muller-10]. It can be seen that the mean is close to zero, and it is a good approximation for the center of the data, i.e., the errors are symmetrical. The variance value indicates that the relative errors for different products are close to each other. In addition to the good statistical parameters, it can be

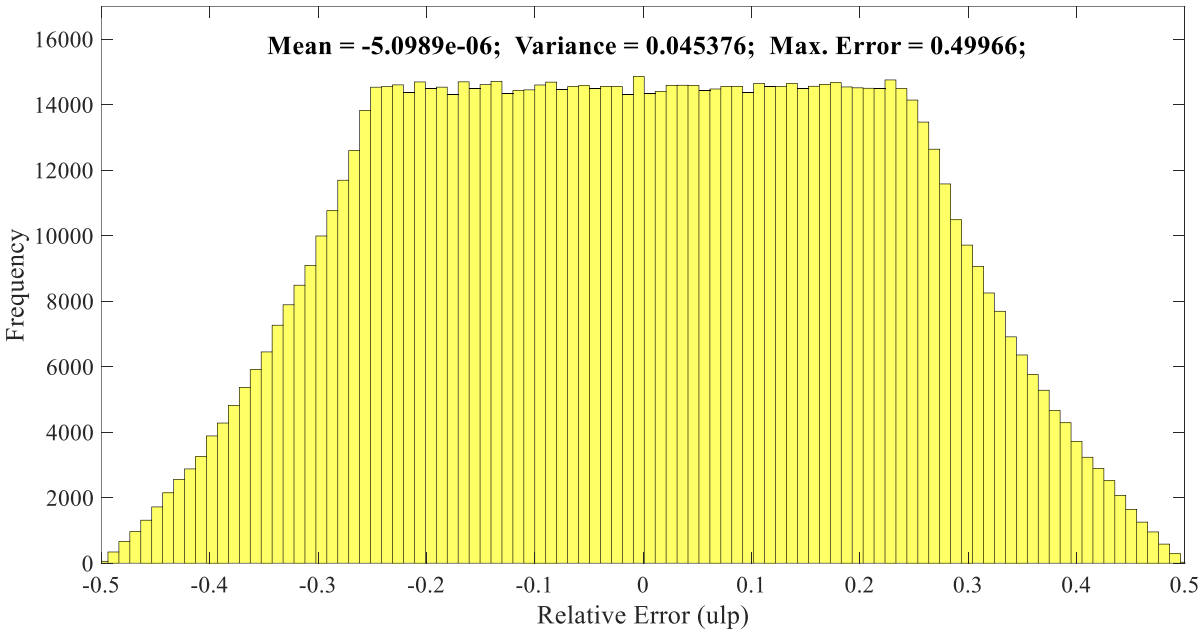

Fig. 4.8 Relative error histogram of the 23-bit FP multiplier with respect to the corresponding double-precision calculated values.

seen that the maximum error is less than ½ ulp, i.e., $e_r \leq 3.812065\times10^{-6}$.

The IEEE 754-2008 standard establishes that IEEE-compliant multipliers must produce results with maximum errors of ½ ulp. The relative error corresponding to ½ ulp for a binary FP format with a precision of *p* bits is defined [Goldberg-91] by

$$2^{-(p+1)} < \frac{1}{2}\text{ulp} \leq 2^{-p}. \tag{4-14}$$

The precision *p* of the 23-bit FP multiplier is 18 bits, therefore, the upper bound evaluation of (4-14) is 2^{-18}, which is greater than the maximum relative error of the proposed design. Therefore, the 23-bit FP multiplier meets the IEEE-754-2008 error specification.

4.4.3.2 Logic-Synthesis Results of the 23-bit FP Multiplier

In order to determine the hardware resources required to implement the 23-bit FP multiplier, the logic synthesis of the multiplier is performed on ASIC technology. The result of the synthesis is a structured netlist based on ARM® standard cells of the Globalfoundries 130 nm CMOS 8RF-DM process. The logic synthesis is performed using the Cadence® RTL Compiler tool.

TABLE 4.8. LOGICAL SYNTHESIS OF THE 23-BIT FP MULTIPLIER ON 130 nm CMOS TECHNOLOGY

Variable	Value
Cell Number	2055
Cell Area	0.030280 mm^2 (49.77 %)
Net Area	0.030555 mm^2 (50.23 %)
Total Area	0.060835 mm^2 (100 %)
Max. Frequency	175.00 MHz
Total Power	13.89 mW

The results of the synthesis are shown in Table 4.8. The total silicon area is 0.060835 mm^2. 49.77% of the total area corresponds to 2,055 standard cells; the remaining area corresponds to the interconnection nets.

The timing optimization of the proposed design is performed to determine the maximum operating frequency. The static timing analysis shows that the multiplier critical-timing path is 5,714 ps, which corresponds to a maximum clock frequency of 175.00 MHz. The total power consumption at this clock frequency is 13.89 mW.

The design and logical synthesis of a 23-bit FP multiplier has been presented. The purpose of the proposed multiplier is to be used in the HF-2cRSR IP core. The specifications of the 23-bit FP multiplier are customized to reduce the hardware requirements and to achieve bit-accurate results of the HF-2cRSR unit. In the next section, the FPGA implementation of the proposed IP core is reported in order to be compared with commercial IP cores from Intel and Xilinx.

4.5. HF-2cRSR Implementation Results and Comparisons

4.5.1 FPGA Implementation

Each block of the HF-2cRSR architecture is modeled using Verilog hardware description language, synthesized on an FPGA technology, and verified using the digital circuit simulator ModelSim. The HF-2cRSR top-level design is a structured Verilog model, where the verified modules of the proposed architecture are instantiated and interconnected as shown in Fig. 4.2 and Fig. 4.3. The functionality of the synthesized top-level design is exhaustively verified using a

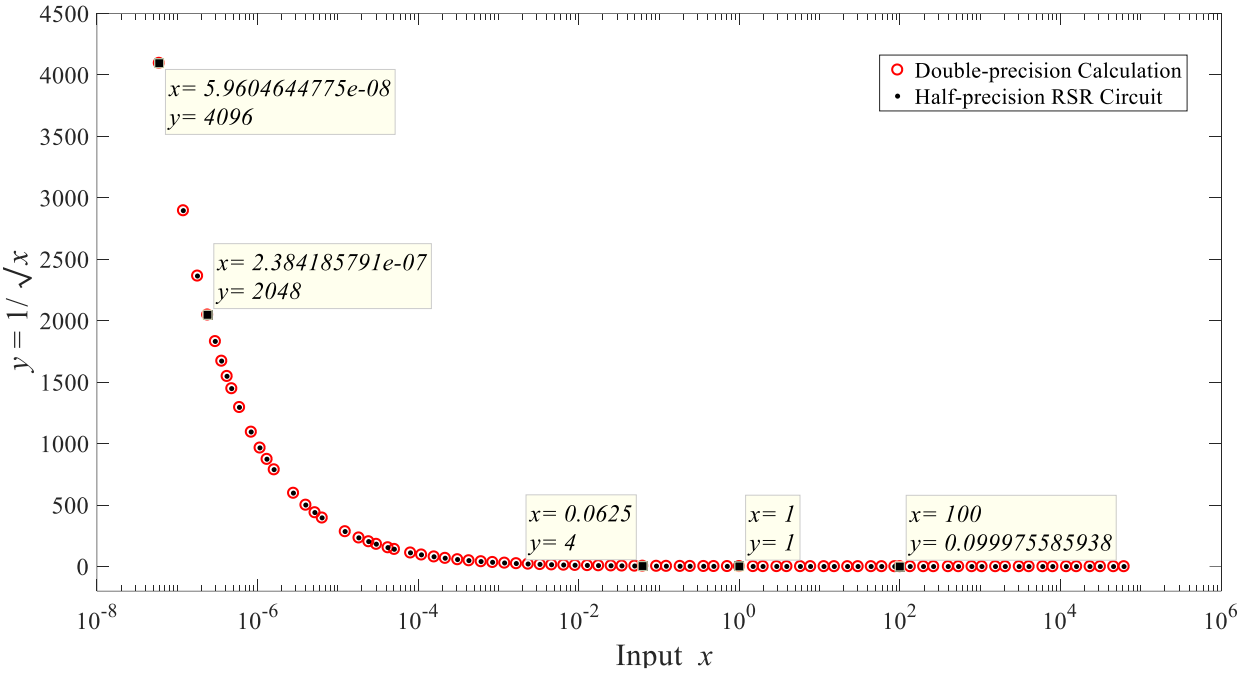

Fig. 4.9 Comparison of the HF-2cRSR outputs with respect to double-precision values.

Verilog test bench, which applies all the possible valid test-vectors (2^{15}) to the unit under test. Since the test-vectors and circuit-simulation results have half-precision FP format, the input-result pairs are exported to a text file as integer numbers to be post-processed in Matlab.

To evaluate the correct functionality and accuracy of the HF-2cRSR IP core, the circuit-level simulation results are compared with the golden values, which are computed evaluating $1/\sqrt{x}$ for all the possible input values using double-precision FP arithmetic and converting the results to half-precision FP bit-patterns using Matlab functions[44]. By means of this comparison, we check that all the circuit simulation results (2^{15}) are exact with respect to the golden values.

A comparison of selected HF-2cRSR results with respect to the corresponding double-precision calculated values, y_{DP}, is shown in Fig. 4.9. As it can be seen, the plot differences are small to be observed at a glance. In order to determine the HF-2cRSR relative error, ε_r, with respect to y_{DP}, the following expression is used

[44] MathWorks, IEEE 754r Half Precision Floating Point Converter. Jul. 19, 2018, https://la.mathworks.com/matlabcentral/fileexchange/23173-ieee-754r-half-precision-floating-point-converter?focused=5133569&tab=function.

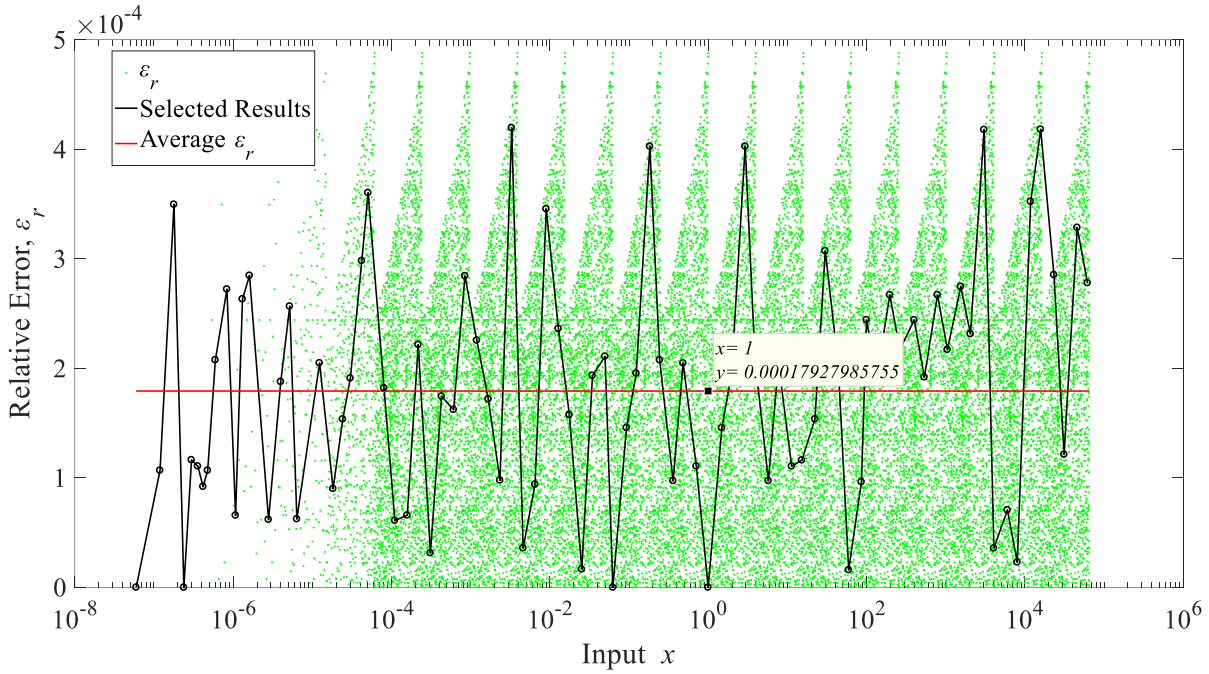

Fig. 4.10 HF-2cRSR relative errors with respect to double-precision FP values.

$$\varepsilon_r = \left| \frac{y - y_{DP}}{y_{DP}} \right|. \tag{4-15}$$

The relative errors for all results provided by the HF-2cRSR circuit with respect to the corresponding y_{DP} values are shown in Fig. 4.10. The relative errors of the selected results are highlighted by the black line; the horizontal line represents the average relative error. The maximum relative error is 4.8768532×10^{-4}, which is lower than ½ ulp of the half-precision format.

With the purpose of comparing the performance of the HF-2cRSR unit versus similar IP cores of Xilinx and Intel, the purposed design is implemented on FPGAs of both companies. The Xilinx and Intel RSR IP cores are available for evaluation in double-precision and single-precision only, therefore, the single-precision version is implemented in this work. The selected FPGAs are Xilinx Artix7 xc7a100t-3csg324 and Intel Stratix 5sgxmb6r3f43c4. The feature size of both devices is 28 nm. The implementations are performed using ISE 14.7 Design Suit and Quartus Prime 16.0 from Xilinx and Intel, respectively. Using the same tools, the single-precision FP IP cores of Xilinx [Xilinx-12] and Intel [Altera-16] for the RSR operation are implemented on their respective FPGAs. The implementation results are shown in Table 4.9 and Table 4.10. The timing of the three post-placed and routed designs are verified by a static-timing analysis performed with

TABLE 4.9. IMPLEMENTATION RESULTS COMPARED WITH INTEL IP CORE
ClkF: clock frequency; C: cycles; Tp: throughput; MOPs: million operations per second

Variable	Intel with DSP	This work with DSP	Intel without DSP	This work without DSP
FP Format	single	half	single	half
FPGA (28 nm)	Stratix V	Stratix V	Stratix V	Stratix V
DSPs (27x27)	5	0	0	0
DSPs (18x18)	1	4	0	0
LUTs	449	658	2884	2028
ClkF (MHz)	122.7	40.76	121.67	33.26
Latency (C)	26	2	26	2
Tp (MOPs)	4.72	20.38	4.68	16.63

the respective companies' tools. The maximum operating frequencies, ClkF, are reported for a slow process (85 °C, 850 mV). The reported throughput (Tp) is calculated dividing the ClkF by the latency of the corresponding implemented IP cores.

4.5.2 Comparison of the HF-2cRSR with Xilinx and Intel IP Cores

For the three IP cores, two types of FPGA implementations are created: using the internal DSP blocks and using only logic. The latter provides a fairer comparison between the commercial IPs and the HF-2cRSR since the number and size of the utilized DSP blocks are very different in the three IP cores. For the implementations using DSP blocks, the HF-2cRSR IP-core utilizes more LUTs than Xilinx and Intel IP cores; this is because many hardware resources of the required multipliers are placed in the DSP blocks. The proposed IP core utilizes fewer and smaller multipliers than the compared implementations. When the implementations are performed without DSP blocks, the HF-2cRSR utilizes 30% fewer LUTs than Intel IP and 50% more than Xilinx IP.

Regarding the throughput of the compared designs, the proposed IP core exhibit 66% more throughput than Xilinx IP and 3.5 times more than Intel IP core. The main advantage of the HF-2cRSR is its low latency, which contributes to better throughput at a low clock frequency, which

TABLE 4.10. IMPLEMENTATION RESULTS COMPARED WITH XILINX IP CORE
ClkF: clock frequency; C: cycles; Tp: throughput; MOPs: million operations per second

Variable	Intel with DSP	This work with DSP	Intel without DSP	This work without DSP
FP Format	single	half	single	half
FPGA (28 nm)	Artix 7	Artix 7	Artix 7	Artix 7
DSPs (25x18)	9	4	0	0
LUTs	366	970	1988	3004
ClkF (MHz)	302.85	33.61	219.68	28.17
Latency (C)	26	2	26	2
Tp (MOPs)	11.65	16.81	8.45	14.08

is an important factor in low-power embedded systems.

4.5.3 Standard-Cell Based Implementation

In order to test the portability of the HF-2cRSR HDL model, and for documenting the required hardware resources for this IP core on ASIC technology, Table 4.11 reports the logic synthesis results of the HF-2cRSR IP core on the 8RF-DM 130nm CMOS technology.

Comparing the implementation results of the two proposed IP cores: the 2C-RSR, and the HF-2cRSR, it can be observed from Table 4.11, that the half-precision floating-point implementation is adequate for applications with limited hardware resources and power consumption. In this case, the HF-2cRSR utilizes 37% less silicon area, and 32% less power consumption than the fixed-point implementation, 2C-RSR. This is at the costs of lower throughput.

4.6. Conclusions

The design and FPGA implementation of a half-precision floating-point RSR IP core has been presented. The design considerations of the modules that make up the HF-2cRSR architecture were discussed. To exemplify the design and verification of FP arithmetic modules a customized

TABLE 4.11. IMPLEMENTATION RESULT COMPARISON OF THE TWO PROPOSED IP CORES

Variable	HF-2cRSR	2C-RSR	Gain
Feature size (nm)	130	130	-
Arithmetic	16-bit FP	16-bit FxP	-
Latency (cycles)	2	2	-
Total Area (mm^2)	0.145189725	0.2289	37%
Max. Frequency (MHz)	25.00 MHz	49.62	-50%
Throughput (MOP)	12.5	24.81	-50%
Total Power (mW)	4.77364	7.06	32%

23-bit FP multiplier is documented, which is one of the fundamental modules of the HF-2cRSR unit. The proposed IP core meets the IEEE 754-2008 accuracy defined for the half-precision RSR operation, it is able to produce a new result in only two clock cycles and the results are accurate at least ½ ulp. The comparisons with respect to Xilinx and Intel IP cores show the impact of half-precision arithmetic on the multiplier size and the advantages of the proposed low-latency design.

The proposed implementation contributes to reduce the lack of half-precision floating-point IP cores, which are suitable options for use in low-precision tolerant applications, such as deep-learning and approximate computing, implemented on low-power embedded systems.

General Conclusions

In this doctoral dissertation, the design and implementation on ASIC technology of two arithmetic units to calculate the reciprocal of the square root (RSR) were presented. The purpose of the implementation was to venture into the development of IP cores on VLSI technology for low-power and low-computational cost embedded applications. The two proposed implementations use an algorithm based on the Newton-Raphson method and on a piecewise-polynomial approximation in a reduced range of the RSR function. The first implementation, 2C-RSR, uses 16-bit fixed-point arithmetic in *Q*(16,11, u) format. The second one, HF-2cRSR, takes advantage of the IEEE 754-2008 half-precision floating-point standard.

In order to carry out the proposed implementations, it was required to have at ITESO integrated circuits laboratory CAD tools and some CMOS technology kit to develop digital ASIC design and to know the design flow of digital integrated circuits using synthesis tools. In Chapter 1 of this doctoral dissertation, the design flow of digital ASICs was presented, some of the Cadence® tools to support this task were described, and the tools that were installed at ITESO integrated circuits laboratory were selected. This enabled laboratory users to perform digital-ASIC designs.

In Chapter 2, the fundamental components for designing digital ASICs were described in detail and a specific CMOS technology (8RF-DM, 130 nm) was selected to be able to send to manufacture the proposed designs. In addition, the workflow and results of the main step of a digital ASIC design front-end, called logic synthesis, was reported, which was implemented for the first time at ITESO integrated circuits laboratory.

In Chapter 3, the design, verification, and physical implementation of a digital integrated circuit that calculates the reciprocal of the square root of a 16-bit fixed-point number were presented. It was proved that all the results of the proposed silicon IP are bit-accurate and have a latency of only two clock cycles. The results of the experimental measurements of the manufactured ASIC on 130 nm CMOS technology were reported. A comparison of the results of the proposed 2C-RSR unit with previously published designs was done; it was shown that the power consumption and latency of the proposed silicon IP are lower. It was then confirmed that these characteristics are suitable for using the 2C-RSR chip in embedded system applications with low-power consumption and low-computational cost.

To evaluate the computational cost and take advantage of the characteristics of the IEEE 754-2008 half-precision floating-point format, in Chapter 4 of this doctoral dissertation the design and implementation of another IP core (HF-2cRSR) that calculates the reciprocal of the square root was documented. The same algorithm was implemented as that one used in the first implementation, however, in this case, 16-bit floating-point arithmetic was used. To illustrate the development of the proposed IP core using this type of arithmetic, the design of a 23-bit floating-point multiplier, tailored to be used in the HF-2cRSR unit, was presented in detail. This second implementation satisfies the accuracy specification defined by the IEEE 754-2008 standard. The HF-2cRSR IP was compared with commercial IP cores from Intel and Xilinx that were implemented on FPGAs. The results of the performed comparisons showed the advantages of the proposed implementation low latency and the positive impact of using half-precision floating-point arithmetic on the required hardware resources and on the word size of the multiplier circuits.

The following research lines are suggested for those interested in giving continuity to this research project or in developing other derived research lines.

The implemented algorithm in the two proposed IP cores uses a 14-segment piecewise-polynomial approximation in a reduced range of the RSR function. In such algorithm, the selection of the number of segments and their location were performed experimentally to achieve the required accuracy. It is suggested, in order to develop a new version of the proposed IP core or for implementing any other IP core, to use function-segmentation methodologies [Trejo-Arellano-17], [Lee-09], for optimizing the number of segments and their location. This could help to obtain a better approximation to the function and a reduction of the required hardware resources to implement it due to the segment number optimization. Furthermore, new methods could be experimented for calculating the seed that the Newton-Raphson method requires, for example, hardware implementations of the magic-number based algorithm [Lomont-03] could be explored, using low-precision floating-point arithmetic with tailored-word size for a specific application.

The IP cores implemented in this project met with the maximum accuracy specification for 16-bit binary arithmetic, which increased the use of hardware resources. However, there are low-precision tolerant applications in the areas of deep learning, fuzzy logic, and encryption algorithms for IoT, among others. It is recommended to identify some specific application in the previously mentioned fields and developing tailored IP cores with the required precision and power consumption by the application. The developed half-precision floating-point modules in this

research project (multiplier, adder, squaring, and RSR) can be modified and completed to create a library of tailored arithmetic IP cores, which can be reused to implement the previously identified low-precision tolerant applications.

It is suggested to explore low-power design techniques such as clock and power-supply gating. This can be added to the implemented workflow at ITESO integrated circuits laboratory with the purpose of applying these techniques to IP cores to be developed.

Another possibility is to develop macromodels based on low-precision floating-point arithmetic functions, using Matlab language or another programming language, to perform analysis of the required precision in algorithms, functions, and modules to be implemented. The macromodels will help quantify the approximation error and determine the optimal-word size for the algorithms or functions to be implemented.

Finally, if an RSR unit with higher working frequency than the implemented IP cores is required, the proposed architecture could be modified to create a pipelined architecture to reduce the critical datapath of the current architecture, which is defined by the multipliers and adders.

Conclusiones Generales

En esta tesis doctoral se presentó el diseño e implementación en tecnología de circuito integrado de aplicación específica (ASIC) de dos unidades aritméticas para calcular el recíproco de la raíz cuadrada (RSR). El propósito de las implementaciones fue incursionar en el desarrollo de núcleos de propiedad intelectual en tecnología VLSI para aplicaciones en sistemas embebidos de baja potencia y bajo costo computacional. Las dos implementaciones propuestas utilizan un algoritmo basado en el método de Newton-Raphson y en una aproximación polinomial por partes en un rango reducido de la función RSR. La primera implementación propuesta, 2C-RSR, utiliza aritmética de punto fijo en un formato de 16 bits, $Q(16,11,u)$. La segunda implementación, HF-2cRSR, aprovecha el estándar IEEE 754-2008 de media precisión de punto flotante.

Para poder realizar las implementaciones propuestas, se requería contar en el laboratorio de circuitos integrados del ITESO con herramientas de diseño asistido por computadora (CAD) y alguna biblioteca de tecnología CMOS para desarrollar el diseño de ASICs digitales y conocer el flujo de diseño de circuitos integrados digitales utilizando herramientas de síntesis. En el Capítulo 1 de esta tesis doctoral se presentó el flujo de diseño de ASICs digitales, se describieron algunas de las herramientas de Cadence® para soportar dicha tarea, y se seleccionaron las herramientas CAD que se instalaron en el laboratorio de circuitos integrados del ITESO; lo anterior habilitó el laboratorio para que los usuarios puedan realizar diseño digital de ASICs.

En el Capítulo 2 se describieron de forma detallada los componentes fundamentales para realizar diseño de ASICs digitales, se seleccionó una tecnología CMOS específica (8RF-DM, 130 nm) para poder fabricar los diseños propuestos. Además, se reportaron el flujo de trabajo y los resultados de la primera fase del flujo de diseño de un ASIC digital, denominada síntesis lógica, la cual fue realizada por primera vez en el laboratorio de circuitos integrados del ITESO.

En el Capítulo 3 se presentó el diseño, verificación, e implementación física de un circuito integrado digital que calcula el recíproco de la raíz cuadrada de un numero de 16 bits en punto fijo. Se comprobó que los 16 bits de todos los resultados de la IP de silicio propuesta son exactos y tienen una latencia de solamente dos ciclos de reloj. Se reportaron los resultados de las mediciones experimentales del ASIC manufacturado en tecnología CMOS de 130 nm. Se realizó una comparación de los resultados del circuito integrado 2C-RSR con diseños publicados previamente

y se demostró que las características de consumo de potencia y latencia del núcleo IP propuesto son menores. Por lo tanto, se confirmó que dichas características son adecuadas para utilizar la IP 2C-RSR en aplicaciones de sistemas embebidos de bajo consumo de potencia y bajo costo computacional.

Para evaluar el costo computacional y aprovechar las características del formato de punto flotante y media precisión que ofrece el estándar 754-2008 del IEEE, en el Capítulo 4 de esta tesis doctoral se documentó el diseño e implementación de otro núcleo IP (HF-2cRSR) que calcula el recíproco de la raíz cuadrada. Se implementó el mismo algoritmo que el utilizado en la primera implementación, sin embargo, en este caso se usó aritmética de punto flotante de 16 bits. Para ejemplificar el desarrollo de la IP propuesta utilizando esté tipo de aritmética, se presentó en forma detallada el diseño de un multiplicador de punto flotante de 23-bits, realizado a la medida para ser utilizado en la unidad HF-2cRSR. Esta segunda implementación cumple con la especificación de exactitud definida por el estándar 754-2008 del IEEE. Se realizó la comparación la IP HF-2cRSR con núcleos IP comerciales de Intel y Xilinx que fueron implementados en FPGAs. Los resultados de las comparaciones realizadas mostraron las ventajas de la baja latencia de la implementación propuesta y el impacto positivo de utilizar aritmética de punto flotante de media precisión en los recursos de hardware requeridos y en el tamaño de palabra de los circuitos multiplicadores.

Enseguida se sugieren algunas líneas de investigación para darle continuidad a este proyecto de investigación, o para emprender otros que se pueden derivar de éste.

El algoritmo implementado en los dos núcleos IP propuestos, utiliza una aproximación polinomial por partes de 14 segmentos en un rango reducido de la función RSR. En el algoritmo implementado, la selección del número de segmentos y su localización se realizó de forma experimental hasta lograr la exactitud requerida. Se propone, para desarrollar una nueva versión los núcleos IP propuestos o para implementar otra función, utilizar metodologías de segmentación de funciones [Trejo-Arellano-17], [Lee-09], para optimizar el número de segmentos y su localización, lo cual puede contribuir a obtener una mejor aproximación a la función y una reducción de recursos de hardware requeridos para implementarla. Esto es debido a la optimización del número de segmentos. También se podría experimentar con nuevos métodos para el cálculo de la semilla que requiere el método de Newton-Raphson, por ejemplo, se podrían explorar realizaciones en hardware del algoritmo basado en el número mágico [Lomont-03], usando aritmética de punto flotante de baja precisión, con tamaño de palabra a la medida de alguna

aplicación específica.

Los núcleos IP implementados en este proyecto cumplen con la máxima especificación de exactitud que se puede lograr con aritmética binaria de 16 bits, lo cual incrementa el uso de recursos de hardware. Sin embargo, existen aplicaciones tolerantes a baja precisión en las áreas de aprendizaje profundo, lógica difusa, y algoritmos de encriptación para IoT, entre otras. Se recomienda identificar alguna aplicación específica en dichos campos y desarrollar los núcleos IP a la medida, con la precisión y consumo de potencia requerida por la aplicación. Los núcleos de punto flotante de media precisión desarrollados en este proyecto de investigación (multiplicador, sumador, elevación al cuadrado, y RSR) pueden ser reutilizados, modificados, y completados para crear una biblioteca de núcleos IP aritméticos, los cuales se puedan reusar para implementar las aplicaciones tolerantes a baja precisión previamente identificadas.

También se sugiere explorar técnicas de diseño para bajo consumo de potencia, por ejemplo, apagado de señal de reloj y fuente de alimentación, las cuales se puede agregar al flujo de trabajo implementado en el laboratorio de circuitos integrados del ITESO. Lo anterior con el propósito de aplicar dichas técnicas a los núcleos IP a desarrollar.

Otra línea de posible trabajo futuro consiste en desarrollar macromodelos basados en funciones aritméticas de punto flotante y baja precisión, en lenguaje de Matlab o en otro lenguaje de programación, que sirvan para hacer análisis de la precisión requerida en los algoritmos, funciones y módulos que se pretendan implementar. Los macromodelos ayudarían a cuantificar el error de aproximación y a dimensionar el tamaño de palabra óptimo para el algoritmo o función a implementar.

Finalmente, si se requiere una unidad RSR con una frecuencia de trabajo más alta que los núcleos IP implementados, la arquitectura propuesta podría modificarse para crear una arquitectura segmentada para reducir la ruta de datos crítica de la arquitectura actual, que está definida por los multiplicadores y sumadores.

Appendix

A. LIST OF INTERNAL RESEARCH REPORTS

1) C. R. Aguilera-Galicia and O. H. Longoria-Gándara, "Digital integrated circuit design flow using Cadence tools at ITESO," Internal Report *PhDEngScITESO-13-04-R*, ITESO, Tlaquepaque, Mexico, Dec. 2013.
2) C. R. Aguilera-Galicia and O. H. Longoria-Gándara, "Fundamental components for implementing digital VLSI frontend design at ITESO," Internal Report *PhDEngScITESO-14-04-R*, ITESO, Tlaquepaque, Mexico, Aug. 2014.
3) C. R. Aguilera-Galicia and O. H. Longoria-Gándara, "Logical synthesis of a basic sequential circuit using RTL Compiler," Internal Report *PhDEngScITESO-14-10-R*, ITESO, Tlaquepaque, Mexico, Dec. 2014.
4) C. R. Aguilera-Galicia, O. H. Longoria-Gándara, and L. Pizano-Escalante, "Proposal for MOSIS to fabricate a fast bit-accurate reciprocal square root circuit prototype under its educational research program," Internal Report *PhDEngScITESO-14-21-R*, ITESO, Tlaquepaque, Mexico, Dec. 2014.
5) C. R. Aguilera-Galicia and O. H. Longoria-Gándara, "Logical effort method for estimating path delay of synthesized logic circuits using global foundries 8RF-DM 130nm technology," Internal Report *PhDEngScITESO-15-22-R*, ITESO, Tlaquepaque, Mexico, Dec. 2015.
6) C. R. Aguilera-Galicia and O. H. Longoria-Gándara, "Test-vector generator for testing a 16-bit reciprocal square root integrated circuit," Internal Report *PhDEngScITESO-16-33-R*, ITESO, Tlaquepaque, Mexico, Dec. 2016.
7) C. R. Aguilera-Galicia, O. H. Longoria-Gándara, and L. Pizano-Escalante, "A two-cycle bit-accurate fixed-point reciprocal square root algorithm," Internal Report *PhDEngScITESO-17-42-R*, ITESO, Tlaquepaque, Mexico, Dec. 2017.
8) C. R. Aguilera-Galicia, O. H. Longoria-Gándara, and L. Pizano-Escalante, "VLSI architecture of a two-cycle bit-accurate fixed-point reciprocal square root unit," Internal Report *PhDEngScITESO-17-47-R*, ITESO, Tlaquepaque, Mexico, Dec. 2017.
9) C. R. Aguilera-Galicia, O. H. Longoria-Gándara, L. Pizano-Escalante, J. Vazquez-Castillo, and M. Salim-Maza, "On-chip implementation of a low-latency bit-accurate reciprocal square root unit," Internal Report *PhDEngScITESO-18-10-R*, ITESO, Tlaquepaque, Mexico, May 2018.

10) C. R. Aguilera-Galicia, O. H. Longoria-Gándara, O. A. Guzmán-Ramos, and L. Pizano-Escalante, “23-bit floating-point multiplier for a half-precision RSR unit,” Internal Research Report *PhDEngScITESO-18-14-R*, ITESO, Tlaquepaque, Mexico, May 2018.
11) C. R. Aguilera-Galicia, O. H. Longoria-Gándara, O. A. Guzmán-Ramos, and L. Pizano-Escalante, “IEEE-754 half-precision floating-point low-latency reciprocal square root IP-Core,” Internal Research Report *PhDEngScITESO-18-25-R*, ITESO, Tlaquepaque, Mexico, Sep. 2018.

B. LIST OF PUBLICATIONS

B.1. Conference Papers

1) C. R. Aguilera-Galicia, O. Longoria-Gandara, O. A. Guzmán-Ramos, and L. Pizano-Escalante, "IEEE-754 half-precision floating-point low-latency reciprocal square root IP-core," in *IEEE Latin-American Conf. on Communications (LATINCOM-2018)*, Guadalajara, Mexico, Nov. 2018, vol. 1, pp. 1-6. (ISSN: 2330-989X; p-ISBN: 978-1-5386-6755-2; e-ISBN: 978-1-5386-6754-5; DOI: 10.1109/LATINCOM.2018.8613254).
2) C. R. Aguilera-Galicia, O. Longoria-Gandara, and L. Pizano-Escalante, "Half-precision floating-point multiplier IP core based on 130 nm CMOS ASIC technology," in *IEEE Latin-American Conf. on Communications (LATINCOM-2018)*, Guadalajara, Mexico, Nov. 2018, vol. 1, pp. 1-5. (ISSN: 2330-989X; p-ISBN: 978-1-5386-6755-2; e-ISBN: 978-1-5386-6754-5; DOI: 10.1109/LATINCOM.2018.8613231).

B.2. Journal Paper

1) C. R. Aguilera-Galicia, O. Longoria-Gandara, L. Pizano-Escalante, J. Vázquez-Castillo, and M. Salim-Maza, "On-chip implementation of a low-latency bit-accurate reciprocal square root unit," *Integration - the VLSI Journal*, vol. 63, pp. 9-17, Sep. 2018. (ISSN: 0167-9260; published online: 26 May 2018; DOI: 10.1016/j.vlsi.2018.04.016).

C. GLOSSARY

Word or Acronym	Meaning
ADE	Analog Design Environment
AMS	Analog Mixed Signal
ATPG	Automatic Test Pattern Generation
CCOpt	Clock Concurrent Optimization
CDB	Storage format for Cadence 5: Cadence Data Base
CDCs	Clock Domain Crossings
CPF	Common Power Format
CPF	Common Power Format
DFM	Design for Manufacturing
DFY	Design for Yield
DRC	Design Rule Checking
DVFS	Dynamic Voltage and Frequency Scaling
ECO	Engineering Change Order
EDI	Encounter Digital Implementation
GDSII	Is a data base file format for data exchange of IC layout
GFM	Global Focus Mapping
ICFB	Integrated Circuit Front to Back
LOCV	Location base On Chip Variation
LPS	Low Power Synthesis Option (Cadence tool for addressing low power design issues early in the design cycle)
MDP	Module Data Path
MMMC	Multi-Mode Multi Corner
MSMV	Multi Supply Multi Voltage
MSV	Multiple Supply Voltage
MTCMOS	power gating low power techniques
NEQ	Non-equivalence
OA	Storage format for Cadence 6: Open Access
OSCI	Open System C Initiative
OVM	Open Verification Methodology
PPA	Power Performance and Area

PSO	Power Shut Off
QoR	Quality of Results
QoS	Quality of Silicon
RDL	Re-Distribution Layer
SDC	Synopsys Design Compiler
SDR	Segment Representative Design
SKILL	Cadence Scripting environment/language
SMART	Signal Integrity, Manufacturing Aware Routability and Timing Optimization
Spectre	Cadence's SPICE
SPP	Tool to convert spice to Spectre
SSTA	Statistical Static Timing Analysis
TCF	Toggle Count Format: A special input file to low power synthesis in Cadence tool
TNS	Total Negative slack
UVM	Universal Verification Methodology
VIP	Verification IP

D. RTL COMPILER LOGIC-SYNTHESIS SCRIPT

```
#### Template Script for RTL->Gate-Level Flow (generated from RC v12.10-s012_1)
## Cuauhtemoc Aguilera
## ITESO

if {[file exists /proc/cpuinfo]} {
  sh grep "model name" /proc/cpuinfo
  sh grep "cpu MHz"    /proc/cpuinfo
}

puts "Hostname : [info hostname]"

##############################################################################
## A     Presetting Global Variables and Attributes
##############################################################################

## In this template DESIGN is the name of the top level module

set DESIGN bwco

## This design use the Virginia Tech 180nm standard-cell library. Next line defines a frienly name
set my_stdcell_library vtvt_tsmc180.lib

## Next line defines a frienly name for the Viriginia Tech LEF library
set my_lef_library {/home/usuario/Cuauh/BibliotecasSCells/convert/cdb/vtvt_tsmc180_lef/vtvt_tsmc180.lef}

set SYN_EFF medium

###set MAP_EFF medium
set MAP_EFF high
set DATE [clock format [clock seconds] -format "%b%d-%T"]
set _OUTPUTS_PATH outputs_${DATE}
set _REPORTS_PATH reports_${DATE}
set _LOG_PATH logs_${DATE}
##set ET_WORKDIR <ET work directory>

############################################################
## B   Specifying Explicit Serch Path Attributes
############################################################
set_attribute lib_search_path {/home/usuario/Cuauh/BibliotecasSCells/convert/cdb/Synopsys_Libraries/libs} /

set_attribute script_search_path {.} /

set_attribute hdl_search_path {/home/usuario/Cuauh/Cadence/my_designs/bwco} /

##Uncomment and specify machine names to enable super-threading.
##set_attribute super_thread_servers {<machine names>} /

##Default undriven/unconnected setting is 'none'.
##set_attribute hdl_unconnected_input_port_value 0 | 1 | x | none /
##set_attribute hdl_undriven_output_port_value   0 | 1 | x | none /
```

```
##set_attribute hdl_undriven_signal_value       0 | 1 | x | none /
##set_attribute wireload_mode <value> /

## next attribute controls the amount of information RTL produce
## when executing commands. The higher the value, the more verbose the output (0-9)
set_attribute information_level 9 /

################################################################
##  C   Setting the Target Technology Library (Library setup)
################################################################

set_attribute library $my_stdcell_library /

################################################################
## D  Setting the Synthesis Mode
################################################################

## PLE (PHYSICAL LAYOUT ESTIMATOR)

set_attribute lef_library $my_lef_library /

## set_attribute cap_table_file <file> /

##generates <signal>_reg[<bit_width>] format
#set_attribute hdl_array_naming_style %s\[%d\] /

## Turn on TNS, affects global and incr opto
## Forces optimization for all the endpoints (Total Negative Slack)

set_attribute tns_opto true /

#########################################################################
## E  Loading the Design (HDL files)
#########################################################################

set_attribute hdl_language v2001

read_hdl -v2001 {contador_m16.v contador_comparador.v bwco.v }

#########################################################################
## F Performing Elaboration
#########################################################################

elaborate $DESIGN

puts "Runtime & Memory after 'read_hdl'"
timestat Elaboration

## Reports all the information for the design (undriven, multidriven, ports and pins, etc.)
## with a summary at the end

puts "Saving check_design_${DESIGN}_all.txt"
```

```
check_design -all > check_design_${DESIGN}_all.txt

################################################################
## G Applying Constraints
################################################################

## read_sdc <file_name>

read_sdc {/home/usuario/Cuauh/Cadence/my_designs/bwco/ucf_bwco_sdc.sdc}

puts "The number of exceptions is [llength [find /designs/$DESIGN -exception *]]"

## Check syntax
# set_attribute force_wireload <wireload name> "/designs/$DESIGN"
## Forces RTL Compiler to use the specified wire-load model
## auto_select automatically selects wire-load models according to the
## wire-load selection table or default wire-load model in the technology library.

#set_attribute force_wireload [find /designs/$DESIGN]

## set_attribute force_wireload [find [find / -library $my_stdcell_library] \
## -wireload "10x10"] [find / -design $DESIGN]

## Creating reports folders

if {![file exists ${_LOG_PATH}]} {
  file mkdir ${_LOG_PATH}
  puts "Creating directory ${_LOG_PATH}"
}
if {![file exists ${_OUTPUTS_PATH}]} {
  file mkdir ${_OUTPUTS_PATH}
  puts "Creating directory ${_OUTPUTS_PATH}"
}

if {![file exists ${_REPORTS_PATH}]} {
  file mkdir ${_REPORTS_PATH}
  puts "Creating directory ${_REPORTS_PATH}"
}

## report timing -ling:
## Reports, in an abbreviated output, possible timing problems in
## the design. These problems can be caused by generated
## clocks, paths constrained with different clocks, ports that have
## no external delays, primary inputs that have no external driver
## or input transition set, primary outputs without external load,
## timing exceptions that cannot be satisfied, constraints that may
## have no impact on the design, and so on.

puts "####################"
puts "Timing -lint Report"
puts "####################"

report timing -lint
```

```
#################################################################################
## H     Applying Optimization Constraints
## Define cost groups (clock-clock, clock-output, input-clock, input-output)
#################################################################################

## Uncomment to remove already existing costgroups before creating new ones.
## rm [find /designs/* -cost_group *]

puts "Defining Cost Groups"

if {[llength [all::all_seqs]] > 0} {
  define_cost_group -name I2C -design $DESIGN
  define_cost_group -name C2O -design $DESIGN
  define_cost_group -name C2C -design $DESIGN
  path_group -from [all::all_seqs] -to [all::all_seqs] -group C2C -name C2C
  path_group -from [all::all_seqs] -to [all::all_outs] -group C2O -name C2O
  path_group -from [all::all_inps]  -to [all::all_seqs] -group I2C -name I2C
}

define_cost_group -name I2O -design $DESIGN
path_group -from [all::all_inps]  -to [all::all_outs] -group I2O -name I2O
foreach cg [find / -cost_group *] {
  report timing -cost_group [list $cg] >> $_REPORTS_PATH/${DESIGN}_pretim.rpt
}

#### To turn off sequential merging on the design
#### uncomment & use the following attributes:

##set_attribute optimize_merge_flops false /
##set_attribute optimize_merge_latches false /
#### For a particular instance use attribute 'optimize_merge_seqs' to turn off sequential merging.

#################################################################################################
#######
## I Performing Synthesis
## Synthesizing to generic
#################################################################################################
#######

synthesize -to_generic -eff $SYN_EFF
puts "Runtime & Memory after 'synthesize -to_generic'"
timestat GENERIC
report datapath > $_REPORTS_PATH/${DESIGN}_datapath_generic.rpt
generate_reports -outdir $_REPORTS_PATH -tag generic
summary_table -outdir $_REPORTS_PATH

#################################################################################################
#######
## I Performing Synthesis
## Synthesizing to gates
#################################################################################################
#######
```

```
synthesize -to_mapped -eff $MAP_EFF -no_incr
puts "Runtime & Memory after 'synthesize -to_map -no_incr'"
timestat MAPPED
report datapath > $_REPORTS_PATH/${DESIGN}_datapath_map.rpt

foreach cg [find / -cost_group *] {
  report timing -cost_group [list $cg] > $_REPORTS_PATH/${DESIGN}_[basename $cg]_post_map.rpt
}
generate_reports -outdir $_REPORTS_PATH -tag map
summary_table -outdir $_REPORTS_PATH


##Intermediate netlist for LEC verification..
write_hdl -lec > ${_OUTPUTS_PATH}/${DESIGN}_intermediate.v
write_do_lec -revised_design ${_OUTPUTS_PATH}/${DESIGN}_intermediate.v -logfile
${_LOG_PATH}/rtl2intermediate.lec.log > ${_OUTPUTS_PATH}/rtl2intermediate.lec.do

## ungroup -threshold <value>
################################################################################################
#########
## I Performing Synthesis
## Incremental Synthesis
################################################################################################
##########

## Uncomment to remove assigns & insert tiehilo cells during Incremental synthesis:

## Removes assigns statements and replaces with buffer/inverter
##set_attribute remove_assigns true /

## Controls the aspects of the replacement of assign statements in the design with buffers or
## inverters, which is controlled by the remove_assigns root attribute

##set_remove_assign_options -buffer_or_inverter <libcell> -design <design|subdesign>

## Determines whether a constant assignment should be replaced with a tie cell in the netlist
##set_attribute use_tiehilo_for_const <none|duplicate|unique> /

synthesize -to_mapped -eff $MAP_EFF -incr
generate_reports -outdir $_REPORTS_PATH -tag incremental
summary_table -outdir $_REPORTS_PATH

puts "Runtime & Memory after incremental synthesis"
timestat INCREMENTAL

foreach cg [find / -cost_group -null_ok *] {
  report timing -cost_group [list $cg] > $_REPORTS_PATH/${DESIGN}_[basename $cg]_post_incr.rpt
}
################################################
## Spatial mode optimization
################################################
```

```
## Uncomment to enable spatial mode optimization
##synthesize -to_mapped -spatial

#################################################################################################
#########
## write Encounter file set (verilog, SDC, config, etc.)
#################################################################################################
#########
##write_encounter design -basename <path & base filename> -lef <lef_file(s)>

## Reports the critical path slack, total negative slack (TNS), number of gates on the critical path,
## and number of violating paths for each cost group. It also gives the instance count, total area
## (net and cell area), cell area, runtime, and host name information


################################################
## J Reporting Synthesis Results
################################################
report qor > $_REPORTS_PATH/${DESIGN}_qor.rpt
report area > $_REPORTS_PATH/${DESIGN}_area.rpt
report datapath > $_REPORTS_PATH/${DESIGN}_datapath_incr.rpt
report messages > $_REPORTS_PATH/${DESIGN}_messages.rpt
report gates > $_REPORTS_PATH/${DESIGN}_gates.rpt

## Generates all the files needed to reload the session in RTL Compiler (for example, .g, .v.and .tcl files)
## If you want to generate all the files that are need to loaded in both a RTL
## Compiler and Encounter session, use the -encounter option
write_design -basename ${_OUTPUTS_PATH}/${DESIGN}_m -encounter $DESIGN

## Generates one of the following design implementations in Verilog format:
## a) A structural netlist using generic logic
## b) A structural netlist using mapped logic
write_hdl  > ${_OUTPUTS_PATH}/${DESIGN}_m.v

################################################
## K     Writing Out Files for Place and Route Tool
################################################
## Generates a script that contains the timing for all modes and
## the design rule constraints of the design
write_script > ${_OUTPUTS_PATH}/${DESIGN}_m.script

## Writes out the current design constraints in Synopsys Design Constraint (SDC) format
write_sdc > ${_OUTPUTS_PATH}/${DESIGN}_m.sdc

#################################
### write_do_lec
#################################
## Translates RTL Compiler settings to Encounter Conformal Logical Equivalence Checking commands
write_do_lec -golden_design ${_OUTPUTS_PATH}/${DESIGN}_intermediate.v -revised_design
${_OUTPUTS_PATH}/${DESIGN}_m.v -logfile  ${_LOG_PATH}/intermediate2final.lec.log >
${_OUTPUTS_PATH}/intermediate2final.lec.do

##Uncomment if the RTL is to be compared with the final netlist..
```

```
write_do_lec -revised_design ${_OUTPUTS_PATH}/${DESIGN}_m.v -logfile ${_LOG_PATH}/rtl2final.lec.log >
${_OUTPUTS_PATH}/rtl2final.lec.do

puts "Final Runtime & Memory."
timestat FINAL
puts "============================"
puts "Synthesis Finished ........."
puts "============================"

## Copy the rc log to the specifyed path
file copy [get_attr stdout_log /] ${_LOG_PATH}/.

################################
### L Exiting RTL Compiler
################################

## quit
## exit
```

Bibliography

[Aguilera-Galicia-16] C. R. Aguilera-Galicia and O. H. Longoria-Gándara, "Test-vector generator for testing a 16-bit reciprocal square root integrated circuit," Internal Report *PhDEngScITESO-16-33-R*, ITESO, Tlaquepaque, Mexico, Dec. 2016.

[Aguilera-Galicia-18] C. R. Aguilera-Galicia, O. Longoria-Gandara, L. Pizano-Escalante, J. Vázquez-Castillo, and M. Salim-Maza, "On-chip implementation of a low-latency bit-accurate reciprocal square root unit," *Integration, the VLSI Journal*, vol. 63, pp. 9–17, Sep. 2018.

[Altera-16] Altera. (2016). *Floating-Point IP Cores User Guide* [Online]. Available: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug_altfp_mfug.pdf.

[Blaauw-14] D. Blaauw, D. Sylvester, P. Dutta, Y. Lee, I. Lee, S. Bang, P. Pannuto, Y. Kim, G. Kim, Y.-S. Kuo, D. Yoon, W. Jung, Z. Foo, Y.-P. Chen, S. Oh, S. Jeong, and M. Choi, "IoT design space challenges: circuits and systems," in *Symp. on VLSI Technology: Digest of Technical Papers*, Honolulu, HI, USA, Jun. 2014.

[Brunvand-10] E. Brunvand, *Digital VLSI Chip Design with Cadence and Synopsys CAD Tools*. Boston, MA: Addison-Wesley, 2010.

[Brunvand-13] E. Brunvand. (2013). *Converting your Existing Libraries from CDB to OA* [Online]. Available: https://www.cs.utah.edu/cadence/data/cdb2oa.pdf.

[Butts-11] J. A. Butts, P. T. P. Tang, R. O. Dror, and D. E. Shaw, "Radix-8 digit-by-rounding: achieving high-performance reciprocals, square roots, and reciprocal square roots," in *IEEE Symp. Computer Arithmetic*, Tubingen, Germany, Jul. 2011, pp. 149–158.

[Cadence-12a] Cadence. (2012). *RTL Compiler (RC) Cook Book* [Online]. Available: http://community.cadence.com/cadence_blogs_8/b/ld/archive/2013/11/12/rtl-compiler-beginner-s-guides-available-on-cadence-online-support.

[Cadence-12b] Cadence. (2012). *RTL Compiler-Rapid Adoption Kit* [Online]. Available: http://support.cadence.com/.

[Cadence-14a] Cadence. (2014). *Encounter RTL Compiler Synthesis Flow* [Online]. Available: http://support.cadence.com/.

[Cadence-14b] Cadence. (2014). *Setting Constraints and Performing Timing Analysis Using Encounter RTL Compiler* [Online]. Available: http://support.cadence.com/.

[Cadence-15a] Cadence. (2015). *Command Reference for Encounter RTL Compiler* [Online]. Available: http://support.cadence.com/.

[Cadence-15b] Cadence. (2015). *Attribute Reference for Encounter RTL Compiler* [Online]. Available: http://support.cadence.com/.

[Castorena-13] E. Castorena, C. R. Aguilera-Galicia, and E. Martínez-Guerrero, "Tutorial de Encounter Digital Implementation System (EDI)," ITESO, Tlaquepaque, Mexico, Dec. 2013.

[Chen-13] Y.-L. Chen, C.-Z. Zhan, T.-J. Jheng, and A.-Y. Wu, "Reconfigurable adaptive singular value decomposition engine design for high-throughput MIMO-OFDM systems," *IEEE Trans. VLSI Syst.*, vol. 21, no. 4, pp. 747–760, Apr. 2013.

[Cooney-10] M. Cooney. (2010). *Cadence Design Flows* [Online]. Available: http://www.phys.hawaii.edu/~varner/PHYS476_Spr10/Lectures/CadenceTools.pdf.

[Dharwadkar-10] A. Dharwadkar and A. Ashrafi. (2010). *Cadence Tutorial* [Online]. Available: http://jason.sdsu.edu/~ashrafi/PDF/CADENCE_Tutorial.pdf.

[Engel-10] G. Engel. (2010). *RTL Logic Synthesis Tutorial* [Online]. Available: http://www.siue.edu/~gengel/ece484LabMaterial/RTLsynthesisTut.pdf.

[Ercegovac-00] M. D. Ercegovac, T. Lang, J.-M. Muller, and A. Tisserand, "Reciprocation, square root, inverse square root, and some elementary functions using small multipliers," *IEEE Trans. Comput.*, vol. 49, no. 7, pp. 628–637, Jul. 2000.

[Ercegovac-04] M. D. Ercegovac and T. Lang, *Digital Arithmetic*. San Francisco, CA: Morgan Kaufmann Publishers/Elsevier, 2004.

[Ercegovac-05] M. D. Ercegovac, J.-M. Muller, and A. Tisserand, "Simple seed architectures for reciprocal and square root reciprocal," in *IEEE Asilomar Conf. Sign. Syst. Comput.*, Pacific Grove CA, Oct. 2005, pp. 1167–1171.

[Farmer-11] T. Farmer and W. Gibbs. (2011). *Cadence Tutorial: Using Cadence Encounter Digital Implementation System – Automatic Layout Place & Route Tool* [Online]. Available: http://www.seas.gwu.edu/~vlsi/ece128/SPRING/labs_tutorials/lab7_cadence_encounter_place-route.pdf.

[Foley-17] D. Foley and J. Danskin, "Ultra-performance Pascal GPU and NVLink interconnect," *IEEE Micro*, vol. 37, no. 2, pp. 7–17, May 2017.

[Franzon-99] P. Franzon, S. Perelstein, and A. Hurst. (1999). *Tutorial 1 Introduction to ASIC Design Methodology* [Online]. Available: http://www.ece.ncsu.edu/asic/tutorials/tutor1/tutor1.pdf.

[Goldberg-91] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Computing Surveys*, vol. 23, no. 1, pp. 5–48, Mar. 1991.

[Gurkaynak-06] F. K. Gurkaynak. (2006). *ASIC Design Flow, How to Design Your Own Chip* [Online]. Available: http://www-micrel.deis.unibo.it/MPHS/slidecorso0607/class_vlsi.pdf.

[Ho-17] N.-M. Ho and W.-F. Wong, "Exploiting half precision arithmetic in Nvidia GPUs," in *IEEE High Performance Extreme Computing Conf.*, Waltham, MA, USA, Sep. 2017, pp. 1–6.

[IEEE-08] IEEE. (2008). *754-2008 Standard for Floating-Point Arithmetic* [Online]. Available: http://ieeexplore.ieee.org/document/4610935/.

[Joldes-16] M. Joldes, O. Marty, J. M. Muller, and V. Popescu, "Arithmetic algorithms for extended precision using floating-point expansions," *IEEE Trans. Comput.*, vol. 65, no. 4, pp. 1197–1210, Apr. 2016.

[Khirallah-03] C. Khirallah, P. Coulton, Y. Arora, and J. Ruan, "Fixed and floating-point implementation of DS-CDMA system using complete complementary codes under both frequency selective and flat fading channel conditions," in *IEE Colloquium DSP Enabled Radio*, Scotland, UK, Sep. 2003, pp. 1–9.

[Kim-11] J. Kim, H. Choi, S. Yoon, T. Bang, J. Park, C. Jung, and J. Cong, "An 8M polygons/s 3-D graphics SoC with full hardware geometric and rendering engine for mobile applications," *IEEE Trans. on Very Large Scale Integration Systems*, vol. 19, no. 8, pp. 1490–1495, Aug. 2011.

[Kwon-08] T.-J. Kwon and D. Jeff, "Floating-point division and square root implementation using a Taylor-series expansion algorithm," in *IEEE Int. Conf. Electron. Circuits Syst.*, St. Julien's, Malta, Sep. 2008, pp. 702–705.

[Lang-03] T. Lang and E. Antelo, "Radix-4 reciprocal square-root and its combination with division and square root," *IEEE Trans. Comput.*, vol. 52, no. 9, pp. 1100–1114, Sep. 2003.

[Lee-09] D.-U. Lee, R. C. C. Cheung, W. Luk, and J. D. Villasenor, "Hierarchical segmentation for hardware function evaluation," *IEEE Trans. on VLSI Systems*, vol. 17, no. 1, pp. 103–116, Jan. 2009.

[Liu-17a] C. Liu, C. Tang, Z. Xing, L. Yuan, and Y. Zhang, "Hardware architecture based on parallel tiled QRD algorithm for future MIMO systems," *IEEE Trans. VLSI Syst.*, vol. 25, no. 5, pp. 1714–1724, Jan. 2017.

[Liu-17b] C. Liu, Z. Xing, L. Yuan, C. Tang, and Y. Zhang, "A novel architecture to eliminate bottlenecks in a parallel tiled QRD algorithm for future MIMO systems," *IEEE Trans. Circuits Syst. II*, vol. 64, no. 1, pp. 26–30, Jan. 2017.

[Lomont-03] C. Lomont. (2003). *Fast Inverse Square Root* [Online]. Available: http://www.lomont.org/Math/Papers/2003/InvSqrt.pdf.

[Luethi-08] P. Luethi, C. Studer, S. Duetsch, E. Zgraggen, H. Kaeslin, N. Felber, and W. Fichtner, "Gram-Schmidt-based QR decomposition for MIMO detection: VLSI implementation and comparison," in *IEEE Asia Pacific Conf. Circuits Syst.*, Macao, China, Nov. 2008, pp. 830–833.

[Mahapatra-12] C. Mahapatra, S. Mahboob, V. C. M. Leung, and T. Stouraitis, "Fast inverse square root based matrix inverse for MIMO-LTE systems," in *IEEE Int. Conf. Ctrl. Eng. Comm. Technol.*, Shenyang, Liaoning, China, Dec. 2012, pp. 321–324.

[Markovic-06] D. Markovic, B. Nikolic, and R. W. Brodersen, "Power and area efficient VLSI architectures for communication signal processing," *in IEEE Int. Conf. on Communications*, Jun. 2006, vol. 7, pp. 3223–3228.

[Markovic-07] D. Markovic, B. Nikolic, and R. W. Brodersen, "Power and area minimization for multidimensional signal processing," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 4, pp. 922–934, Apr. 2007.

[Martin-Del-Campo-12] F. Martin-Del-Campo, A. Morales-Reyes, R. Perez-Andrade, R. Cumplido, A.-G. Orozco-Lugo, and C. Feregrino, "A multi-cycle fixed point square root module for FPGAs," *IEICE Electron. Express*, vol. 9, no. 11, pp. 971–977, Jun. 2012.

[Menard-05] D. Menard, D. Chillet, and O. Sentieys, "Floating-to-fixed-point conversion for digital signal processors," *EURASIP J. Applied Sign. Process.*, vol. 2006, no. 96421, pp. 1–19, Jul. 2005.

[Mittal-16] S. Mittal, "A survey of techniques for approximate computing," *ACM Computing Surveys*, vol. 48, no. 4, pp. 62:1–33, May 2016.

[Muller-05] J.-M. Muller, *Elementary Functions: Algorithms and Implementation*, 2nd ed. Boston, MA: Birkhäuser, 2005.

[Muller-10] J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lef`evre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres, *Handbook of Floating-Point Arithmetic*. New York, NY: Birkhäuser Boston, 2010.

[Parra-Michel-18] R. Parra-Michel (CINVESTAV), J. Luis Pizano-Escalante (ITESO), J. Vázquez-Castillo (UQRO), and O. H. Longoria-Gándara (ITESO), "Cálculo Rápido del Recíproco de la Raíz Cuadrada en Punto Fijo," Mexican Patent Application MX/a/2015/007203 (IMPI), June 8, 2015. TÍTULO DE PATENTE No. 354623, March 8, 2018.

[Patel-08a] C. Patel. (2008). *Advanced VLSI Design-Abstract Generation* [Online]. Available: http://www.csee.umbc.edu/~cpatel2/links/641/slides/lect03_abstract.pdf.

[Patel-08b] C. Patel (2008). *Advanced VLSI Design-Standard Cell Library/Library Exchange Format (LEF)* [Online]. Available: http://www.csee.umbc.edu/~cpatel2/links/641/slides/lect04_LEF.pdf.

[Patel-08c] C. Patel. (2008). *Advanced VLSI Design-Liberty Timing File (LIB)* [Online]. Available: http://www.csee.umbc.edu/~cpatel2/links/641/slides/lect05_LIB.pdf.

[Patel-08d] C. Patel. (2008). *Advanced VLSI Design-Standard Cell Design* [Online]. Available: http://www.csee.umbc.edu/~cpatel2/links/641/slides/lect02_std_cells.pdf.

[Peercy-00] M. S. Peercy, M. Olano, J. Airey, and P. J. Ungar, "Interactive multi-pass programmable shading," in *ACM. Int. Conf. Computer Graphics Interactive Techniques*, LA, Jul. 2000, pp. 425–432.

[Piñeiro-02] J.-A. Piñeiro and J. D. Bruguera, "High-speed double-precision computation of reciprocal, division, square root, and inverse square root," *IEEE Trans. Comput.*, vol. 51, no. 12, pp. 1377–1388, Dec. 2002.

[Pizano-Escalante-15] L. Pizano-Escalante, R. Parra-Michel, J. Vazquez-Castillo, and O. Longoria-Gandara, "Fast bit-accurate reciprocal square root," *Elsevier Microprocessors and Microsystems*, vol. 39, no. 2, pp. 74–82, Mar. 2015.

[Ren-93] H. Ren, L. B. Hoang, H.-C. Chen, and B. W. Y. Wei, "Design of a 16-bit CMOS divider/square-root circuit," in *IEEE Asilomar Conf. Sign. Syst. Comput.*, Pacific Grove CA, Nov. 1993, pp. 807–811.

[Rounioja-03] K. Rounioja and J. A. Parviainen, "Arithmetic processing unit for reciprocal operations," in *Int. Symp. on System-on-Chip*, Tampere, Finland, Nov. 2003, pp. 109–112.

[Sajid-12] I. Sajid, M. M. Ahmed, and S. G. Ziavras, "Novel pipelined architecture for efficient evaluation of the square root using a modified non-restoring algorithm," *J. Sign. Process. Syst.*, vol. 67, no. 2, pp. 157–166, May 2012.

[Salmela-06] P. Salmela, A. Happonen, T. Järvinen, A. Burian, and J. Takala, "DSP implementation of Cholesky decomposition," in *IEEE Symp. Trends in Comm.*, Bratislava, Slovakia, Jun. 2006, pp. 6–9.

[Salmela-11] P. Salmela, B. Adrian, T. Järvinen, H. Aki, and J. Takala, "Low-complexity inverse square root approximation for baseband matrix operations," *Hindawi ISRN Signal Processing*, vol. 2011, no. 615934, pp. 1–8, Jan. 2011.

[Schaffer-98] T. Schaffer, A. Stanaski, A. Glaser, and P. Franzon, "The NCSU design kit for IC fabrication through MOSIS," in *Int. Cadence User Group Conf.*, Austin, TX, Sep. 1998, pp. 1–9.

[Schulte-99] M. J. Schulte and K. E. Wires, "High-speed inverse square roots," in *IEEE Symp. Computer Arithmetic*, Adelaide, Australia, Apr. 1999, pp. 124–131.

[Seth-11] A. Seth and W.-S. Gan, "Fixed-point square roots using L-b truncation [DSP tips and tricks]," *IEEE Signal Process. Mag.*, vol. 28, no. 6, pp. 149–1453, Nov. 2011.

[Shan-08] Z. Shan. (2008). *Tutorial PnR: Placement and Routing for a Schematic* [Online]. Available: http://www.egr.msu.edu/classes/ece410/mason/files/Tutorial%20PnR-sp08.pdf.

[Singh-07] C. K. Singh, S. H. Prasad, and P. T. Balsara, "VLSI architecture for matrix inversion using modified Gram-Schmidt based QR decomposition," in *IEEE Int. Conf. VLSI Design-Int. Conf. Embedded Systems*, Bangalore, India, Jan. 2007, pp. 836–841.

[Sohn-06] J.-H. Sohn, J.-H. Woo, M.-W. Lee, H.-J. Kim, R. Woo, and H.-J. Yoo, "A 155-mW 50-M vertices/s graphics processor with fixed-point programmable vertex shader for mobile applications," *IEEE J. Solid-State Circuits*, vol. 41, no. 5, pp. 1081–1091, May 2006.

[Srinivas-95] H. R. Srinivas and K. K. Parhi, "A floating point radix 2 shared division/square root chip," in *IEEE Int. Conf. Computer Design: VLSI in Computers and Processors*, Austin, TX, Oct. 1995, pp. 472–478.

[Sulistyo-10] J. Sulistyo. (2010). *Development of CMOS Standard Cell Library* [Online]. Available: http://www.ece.unm.edu/~jimp/vlsi_synthesis/contrib/vt_std_cells.pdf.

[Sung-95] W. Sung and K. Kum, "Simulation-based word-length optimization method for fixed-point digital signal processing systems," *IEEE Trans. Signal Process.*, vol. 43, no. 12, pp. 3087–3090, Dec. 1995.

[Suresh-13] S. Suresh, S. F. Beldianu, and S. G. Ziavras, "FPGA and ASIC square root design for high performance and power efficiency," in *IEEE Int. Conf. Application-Specific Syst. Architectures Processors*, Washington, DC, Jun. 2013, pp. 269–272.

[Takagi-01] N. Takagi, "A hardware algorithm for computing reciprocal square root," in *IEEE Symp. Computer Arithmetic*, Vail, CO, USA, Jun. 2001, pp. 94–100.

[Theocharides-05] T. Theocharides. (2005). *ASIC Design Flow P&R Tutorial* [Online]. Available: http://www.cse.psu.edu/~cg577/LECTURES/L4-2005.pdf.

[Trejo-Arellano-17] J. M. Trejo-Arellano, *Adaptive Function Segmentation Methodology for Resources Optimization of Hardware-Based Function Evaluators*, Master's Thesis, Dept. of Electronics, Systems and Informatics, ITESO, Tlaquepaque, Mexico, 2017.

[Venkatesh-17] G. Venkatesh, E. Nurvitadhi, and D. Marr, "Accelerating deep convolutional networks using low-precision and sparsity," in *IEEE Int. Conf. Acoustics, Speech and Signal Processing*, New Orleans, LA, Mar. 2017, pp. 2861–2865.

[Wang-09] L. T. Wang, Y. W. Chang, and K. T. Cheng, *Electronic Design Automation: Synthesis, Verification, and Test*. Burlington, MA: Morgan Kaufmann Publishers/Elsevier, 2009.

[Wang-10] D. Wang, M. D. Ercegovac, and N. Zheng, "Design of high-throughput fixed-point complex reciprocal/square-root unit," *IEEE Trans. Circuits Syst. II*, vol. 57, no. 8, pp. 627–631, Aug. 2010.

[Weste-11] N. H. E. Weste and D. M. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th ed. Boston, MA: Addison Wesley, 2011.

[Williams-16] J. H. Williams, *Quantifying Measurement: The Tyranny of Numbers*. San Rafael, CA: Morgan & Claypool, 2016.

[Wires-06] K. E. Wires and M. J. Schulte, "Reciprocal and reciprocal square root units with operand modification and multiplication," *J. VLSI Sign. Process. Syst. Sign. Image Video Technol.*, vol. 42, no. 3, pp. 257–272, Mar. 2006.

[Wong-94] W. F. Wong and E. Goto, "Fast hardware-based algorithms for elementary function computations using rectangular multipliers," *IEEE Trans. Comput.*, vol. 43, no. 3, pp. 278–294, Mar. 1994.

[Woo-09] J.-H. Woo, J.-H. Sohn, H. Kim, and H.-J. Yoo, "A low-power multimedia SoC with fully programmable 3D graphics for mobile devices," *IEEE Comput. Grap. Appl.*, vol. 29, no. 5, pp. 82–90, Sep. 2009.

[Xilinx-12] Xilinx. (2012). *LogiCORE IP Floating-Point Operator* [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/floating_point/v6_1/pg060-floating-point.pdf.

[Yin-16] P. Yin, C. Wang, W. Liu, and F. Lombardi, "Design and performance evaluation of approximate floating-point multipliers," in *IEEE Comp. Society Annual Symp. on VLSI*, Pittsburgh, PA, USA, Jul. 2016, pp. 296–301.

Author Index

Subject Index

S

T

U

V

W

X